

README.md:

cover: .gitbook/assets/HH-Eco-Hero-Desktop-R1.webp

coverY: -16.77342463378693

layout:

cover:

visible: true

size: full

title:

visible: true

description:

visible: false

tableOfContents:

visible: true

outline:

visible: true

pagination:

visible: true

Welcome to Hedera – let's build the future

```
<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><a href="#learn-the-basics"><strong>LEARN THE BASICS</strong></a></td><td align="center"><a href="#learn-the-basics">#learn-the-basics</a></td></tr><tr><td align="center"><a href="#set-up-your-environment"><strong>ENVIRONMENT SETUP</strong></a></td><td align="center"><a href="#set-up-your-environment">#set-up-your-environment</a></td></tr><tr><td align="center"><a href="#build-your-first-decentralized-application-dapp"><strong>START BUILDING</strong></a></td><td align="center"><a href="#build-your-first-decentralized-application-dapp">#build-your-first-decentralized-application-dapp</a></td></tr><tr><td align="center"><a href="#hedera-network-services"><strong>NETWORK SERVICES</strong></a></td><td align="center"><a href="#hedera-network-services">#hedera-network-services</a></td></tr><tr><td align="center"><a href="#evm-compatible-tools"><strong>EVM-COMPATIBLE TOOLS</strong></a></td><td align="center"><a href="#evm-compatible-tools">#evm-compatible-tools</a></td></tr><tr><td align="center"><a href="#developer-tools-and-integrations"><strong>TOOLS &#x26; INTEGRATIONS</strong></a></td><td align="center"><a href="#developer-tools-and-integrations">#developer-tools-and-integrations</a></td></tr><tr><td align="center"><a href="https://hedera.com/roadmap"><strong>HEDERA ROADMAP</strong></a></td><td align="center"><a href="https://hedera.com/roadmap">https://hedera.com/roadmap</a></td></tr><tr><td align="center"><a href="support-and-community/contributing-guide.md"><strong>CONTRIBUTING GUIDE</strong></a></td><td align="center"><a href="support-and-community/contributing-guide.md">contributing-guide.md</a></td></tr></tbody></table>
```

Learn the Basics

Start your journey with the public Hedera network by learning the basics – from understanding the network's architecture to who's building next-generation applications, you'll have a proper foundation to start building.

```
<table data-card-size="large" data-view="cards"><thead><tr><th></th><th></th><th data-hidden data-card-target data-type="content-ref"></th><th data-hidden data-card-cover data-type="files"></th></tr></thead><tbody><tr><td><a href="https://hedera.com/learning/hedera-hashgraph/what-is-hedera-hashgraph"><strong>Hedera Explained</strong></a></td><td>Understand the basics of Hedera network node types, developer services &#x26; API, governance, $HBAR,
```



```
{% tab title="Video Tutorial" %}
{% embed url="https://youtu.be/Skx6b8uK9ks" %}
Start Developing on Hedera\
by Developer Advocate: Michiel Mulders
{% endembed %}
{% endtab %}
{% endtabs %}
```

Build Your First Decentralized Application (DApp)

Learn how to use Hedera network services and build your first dApp by following these step-by-step tutorials.

Create & Deploy a Smart Contract	Create a Solidity Smart Contract that interacts with the Hedera Token Service, bringing full programmability into your token-based application.	deploy-a-contract-using-the-hedera-token-service.md
Create Fungible Tokens	Service to create fungible tokens that map to ERC-20 standards and scale to 10,000+ TPS.	create-and-transfer-your-first-fungible-token.md
ERC-20 standards		ERC-20 standards
create-and-transfer-your-first-fungible-token.md		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards
ERC-721 standards		ERC-721 standards

network.	token-service	token-service-icon.png
	Consensus Service	
Incorporate scalable, real-time, auditable logs of events with consensus timestamps into any web2 or permissionless web3 application.		
	consensus-service	consensus-service-icon.png
	File Service	
Hedera File Service is a decentralized file storage platform that allows users to securely store and access files on a distributed network of computers using hash as a file identifier.		
	readme-1-1	file-service-icon.png

EVM-Compatible Tools

Continue utilizing familiar Ethereum development tools such as Web3.js, Truffle, Ethers, Hardhat, and Foundry to build on Hedera using the JSON-RPC Relay. As an Ethereum developer, your workflow does not have to change.

https://github.com/hashgraph/hedera-json-rpc-relay/blob/main/docs/examples/account-alias-ecdsa.js	Create an ECDSA Account	
Create an ECDSA account that works with JSON-RPC to use EVM-compatible tooling, libraries, and wallets.		
https://github.com/hashgraph/hedera-json-rpc-relay/blob/main/docs/examples/account-alias-ecdsa.js	create ECDSA account.png	
https://github.com/hashgraph/hedera-json-rpc-relay/blob/main/docs/transfer-guide.md	Set Up Metamask	
Add a Hedera network to Metamask using the Hashio implementation of the JSON-RPC Relay. Sign transactions and queries to test your Hedera-powered application.		
https://github.com/hashgraph/hedera-json-rpc-relay/blob/main/docs/transfer-guide.md	setup metamask.png	
https://github.com/hashgraph/hedera-json-rpc-relay/tree/main/tools	Libraries & Tooling	
Integrate Hedera into Ethers, web3.js, Truffle, or Hardhat for building, debugging, and deploying your smart contract applications on the Hedera network.		
https://github.com/hashgraph/hedera-json-rpc-relay/tree/main/tools	libraries & tooling.png	
core-concepts/smart-contracts/deploying-smart-contracts/json-rpc-relay.md	JSON-RPC Relay	
JSON-RPC makes it easy for existing EVM-based libraries, dev tools, and wallets to work effortlessly on Hedera.		
core-concepts/smart-contracts/deploying-smart-contracts/json-rpc-relay.md	JSON-RPC relay.png	

Developer Tools and Integrations

Explore various developer tools and resources on Hedera that help make building and maintaining your decentralized application even easier.

|--|--|--|

target data-type="content-ref"></th><th data-hidden data-card-cover data-type="files"></th></tr></thead><tbody><tr><td>Network Explorers</td><td>Visualizations & APIs for historical network data, including account & token details, transactions, tokens, contracts, topics, and schedules.</td></tr><tr><td>landing-icon-network-explorers.png</td><td>Wallet Integrations</td></tr><tr><td>Onboard users into your application with ease. Wallets on Hedera offer transaction signing, management of HBAR, NFTs, and fungible tokens.</td><td>wallet integrations.png</td></tr><tr><td>Network Bridge</td><td>HashPort is a public utility that facilitates the bi-directional movement of digital assets between public networks in a quick, secure, and cost-effective way.</td></tr><tr><td>network bridge (1).png</td><td>Monitoring & Security</td></tr><tr><td>Use Open Zeppelin's Defender Admin, Relay, and Sentinels for administrative, security, and monitoring functionality for Smart Contracts on Hedera.</td><td>monitor & security.png</td></tr><tr><td>Local Node</td><td>Learn how to set up your own Hedera local network, including consensus & mirror nodes, using Docker.</td></tr><tr><td>local node (1).png</td><td>Local Node</td></tr></tbody></table>
--

> Have a question? Ask it on StackOverflow.

SUMMARY.md:

Table of contents

Welcome to Hedera – let's build the future

Getting Started

Get Your Testnet Account

Environment Setup

Create an Account

Transfer HBAR

Query Ledger Data

Tutorials

Hello World

Create and fund account

HTS: Fungible Token

HSCS: Smart Contract

HCS: Topic

Smart Contracts

How to Verify a Smart Contract on HashScan

Deploy a Smart Contract Using Remix

Deploy a Smart Contract Using Hardhat and Hedera JSON-RPC Relay

Deploy Your First Smart Contract

Deploy a Contract Using the Hedera Token Service

Send and Receive HBAR Using Solidity Smart Contracts

Create an HBAR Faucet App Using React and MetaMask

- Deploy By Leveraging Ethereum Developer Tools On Hedera
- Deploy a Subgraph Using The Graph and Hedera JSON-RPC Relay
- Deploy Smart Contracts on Hedera Using Truffle
- The Power of Native Hedera Tokens as ERC-20 Tokens: A step-by-step guide
- Hedera Smart Contracts Workshop
 - Setup
 - Solidity
 - Hedera SDK JS
 - Hardhat and EthersJs
 - Outro
- Foundry
 - How to Setup Foundry and Write a Basic Unit Test
 - How to Deploy and Verify a Hedera Smart Contract with Foundry
 - How to Test A Solidity Event
 - How to Fork Testnet on Latest Block
- Consensus
 - Submit Your First Message
 - Submit Message to Private Topic
 - Query Messages with Mirror Node
- Tokens
 - Create and Transfer Your First NFT
 - Create and Transfer Your First Fungible Token
 - Create and Transfer an NFT using a Solidity Contract
 - Structure Your Token Metadata Using JSON Schema V2
 - Hedera Token Service - Part 1: How to Mint NFTs
 - Hedera Token Service - Part 2: KYC, Update, and Scheduled Transactions
 - Hedera Token Service - Part 3: How to Pause, Freeze, Wipe, and Delete NFTs
- Local Node
 - How to Run Hedera Local Node in a Cloud Development Environment (CDE)
 - Run a Local Node in Gitpod
 - Run a Local Node in Codespaces
 - How to Set Up a Hedera Local Node
 - Set Up a Hedera Local Node using the NPM CLI
- More Tutorials
 - How to Create a Personal Access Token (API Key) on the Hedera Portal
 - How to Auto-Create Hedera Accounts with HBAR and Token Transfers
 - How to Configure a Mirror Node and Query Data
 - How to Generate a Random Number on Hedera
 - Get Started with the Hedera Consensus Service Fabric Plugin
 - Virtual Environment Setup
 - Schedule Your First Transaction
 - How to Connect to Hedera Networks Over RPC
 - Configuring Hashio RPC endpoints
 - Configuring Hedera JSON-RPC Relay endpoints
 - Configuring Validation Cloud RPC endpoints
 - JavaScript Testing
 - Create a Hedera DApp Integrated with WalletConnect
 - How to Connect MetaMask to Hedera
- Demo Applications
- Starter Projects
- Building on Hedera (course)
- Networks
 - Mainnet
 - Mainnet Accounts
 - Mainnet Consensus Nodes
 - Node Requirements
 - FAQ
 - Fees
 - Transaction Records
- Testnets
 - Testnet Accounts
 - Testnet Consensus Nodes
- Localnet
 - Single Node Configuration

- Multinode Configuration
- Network Explorers and Tools
- Release Notes
 - Hedera Services
 - Hedera Mirror Node
- Core Concepts
 - Accounts
 - Account Creation
 - Auto Account Creation
 - Account Properties
 - Keys and Signatures
 - Schedule Transaction
 - Smart Contracts
 - Understanding Hedera for EVM Developers
 - Creating Smart Contracts
 - Compiling Smart Contracts
 - Deploying Smart Contracts
 - Gas and Fees
 - JSON-RPC Relay
 - Smart Contract Addresses
 - Verifying Smart Contracts
 - Smart Contract Traceability
 - Tokens Managed by Smart Contracts
 - ERC-20 (Fungible Tokens)
 - ERC-721: Non-Fungible Tokens (NFTs)
 - Hedera Token Service System Contract
 - Smart Contract Rent
 - Smart Contract Security
 - Staking
 - Staking Program
 - Stake HBAR
 - Hashgraph Consensus Algorithm
 - Gossip About Gossip
 - Virtual Voting
 - Transactions and Queries
 - State and History
 - Mirror Nodes
 - Hedera Mirror Node
 - One Click Mirror Node Deployment
 - Run Your Own Mirror Node
 - Run Your Own Mirror Node with Google Cloud Storage (GCS)
 - Run Your Mirror Node with Amazon Web Services S3 (AWS)
- Open Source Solutions and Integrations
 - HashioDAO
 - Governance Token DAO
 - NFT DAO
 - Multisig DAO
 - DAO Proposals
 - Local Environment Setup
 - Hedera Custodians Library
 - How to use it
 - Hedera Wallet Snap By MetaMask
 - Hedera Wallet Snap Documentation
 - Tutorial: MetaMask Snaps – What Are They and How to Use Them
 - Oracle Networks
 - Pyth Oracles
 - Supra Oracles
 - Stablecoin Studio
 - Core Concepts
 - Web UI Application
 - CLI Management
 - TypeScript SDK
 - Hedera Guardian
 - Hedera WalletConnect

SDKs & APIs

SDKs

- Build Your Hedera Client
- Set Up Your Local Network
- Network Address Book

Keys

- Generate a new key pair
- Import an existing key
- Create a key list
- Create a threshold key
- Generate a mnemonic phrase
- Recover keys from a mnemonic phrase

HBAR

Specialized Types

- Pseudorandom Number Generator

Transactions

- Transaction ID
- Modify transaction fields
- Create an unsigned transaction
- Manually sign a transaction
- Submit a transaction
- Sign a multisignature transaction
- Get a transaction receipt
- Get a transaction record

Schedule Transaction

- Schedule ID
- Create a scheduled transaction
- Sign a scheduled transaction
- Delete a scheduled transaction
- Get schedule info
- Network Response Messages

Queries

- General Network Response Messages

Accounts and HBAR

- Create an account
- Update an account
- Transfer cryptocurrency
- Approve an allowance
- Delete an allowance
- Delete an account
- Get account balance
- Get account info
- Network Response Messages

Consensus Service

- Create a topic
- Update a topic
- Submit a message
- Delete a topic
- Get topic messages
- Get topic info
- Network Response

Token Service

- Token ID
- NFT ID
- Token types
- Create a token
- Custom token fees
- Update a token
- Update token custom fees
- Update NFT metadata
- Transfer tokens
- Reject an airdrop
- Delete a token
- Mint a token

- Burn a token
- Freeze an account
- Unfreeze an account
- Enable KYC account flag
- Disable KYC account flag
- Associate tokens to an account
- Dissociate tokens from an account
- Pause a token
- Unpause a token
- Wipe a token
- Atomic swaps
- Get account token balance
- Get token info
- Get NFT info
- Network Response Messages
- File Service
 - Create a file
 - Append to a file
 - Update a file
 - Delete a file
 - Get file contents
 - Get file info
 - Network Response Messages
- Smart Contract Service
 - Delegate Contract ID
 - Create a smart contract
 - Update a smart contract
 - Delete a smart contract
 - Call a smart contract function
 - Ethereum transaction
 - Get a smart contract function
 - Get smart contract bytecode
 - Get smart contract info
 - Hedera Service Solidity Libraries
 - Network Response Messages
- Signature Provider
 - Provider
 - Signer
 - Wallet
 - Local Provider
- REST API
- Hedera Consensus Service gRPC API
- Hedera APIs
 - Basic Types
 - AccountAmount
 - AccountID
 - ContractID
 - CryptoAllowance
 - CurrentAndNextFeeSchedule
 - FeeComponents
 - FeeData
 - FeeSchedule
 - FileID
 - Fraction
 - HederaFunctionality
 - Key
 - KeyList
 - NftAllowance
 - NftTransfer
 - NodeAddress
 - NodeAddressBook
 - RealmID
 - ScheduleID
 - SemanticVersion

- ServicesConfigurationList
- ServiceEndpoint
- Setting
- ShardID
- Signature
- SignatureList
- SignatureMap
- SignaturePair
- SubType
- TransferList
- TransactionID
- ThresholdKey
- ThresholdSignature
- TokenAllowance
- TokenBalance
- TokenBalances
- TokenFreezeStatus
- TokenPauseStatus
- TokenID
- TokenKycStatus
- TokenRelationship
- TokenTransferList
- TokenType
- TokenSupplyType
- TopicID
- TransactionFeeSchedule
- Cryptocurrency Accounts
 - CryptoService
 - CryptApproveAllowance
 - CryptoDeleteAllowance
 - CryptoCreate
 - CryptoTransfer
 - CryptoUpdate
 - CryptoDelete
 - CryptoGetAccountBalance
 - CryptoGetAccountRecords
 - CryptoGetInfo
 - CryptoGetStakers
- Consensus Service
 - Consensus Service
 - ConsensusCreateTopic
 - ConsensusUpdateTopic
 - ConsensusSubmitMessage
 - ConsensusDeleteTopic
 - ConsensusTopicInfo
 - ConsensusGetTopicInfo
- Schedule Service
 - ScheduleService
 - SchedulableTransactionBody
 - ScheduleCreate
 - ScheduleDelete
 - ScheduleSign
 - ScheduleGetInfo
- Token Service
 - TokenService
 - CustomFees
 - AssessedCustomFee
 - CustomFee
 - FractionalFee
 - FixedFee
 - RoyaltyFee
 - TokenCreate
 - TokenUpdate
 - TokenFeeScheduleUpdate

- TokenDelete
- TokenMint
- TokenBurn
- TokenFreezeAccount
- TokenUnfreezeAccount
- TokenGrantKyc
- TokenRevokeKyc
- TokenAssociate
- TokenDissociate
- TokenWipeAccount
- TokenPause
- TokenUnpause
- TokenGetInfo
- TokenGetNftInfo
- TokenGetNftInfos
- TokenGetAccountNftInfo
- File Service
 - FileService
 - FileCreate
 - FileAppend
 - FileUpdate
 - FileDelete
 - FileGetContents
 - FileGetInfo
- Smart Contracts
 - SmartContractService
 - ContractCall
 - ContractCallLocal
 - ContractCreate
 - ContractUpdate
 - ContractDelete
 - ContractGetByteCode
 - ContractGetInfo
 - ContractGetRecords
- Miscellaneous
 - Duration
 - ExchangeRate
 - Freeze
 - FreezeType
 - GetByKey
 - GetBySolidityID
 - NetworkGetVersionInfo
 - NetworkService
 - Query
 - QueryHeader
 - Response
 - ResponseCode
 - ResponseHeader
 - SystemDelete
 - SystemUndelete
 - TimeStamp
 - Transaction
 - TransactionBody
 - TransactionContents
 - TransactionGetFastRecord
 - TransactionGetReceipt
 - TransactionGetRecord
 - TransactionReceipt
 - TransactionRecord
 - TransactionResponse
 - UncheckedSubmit
- Deprecated
 - SDKs (V1)
 - Build your Hedera client

- Set-up Your Local Network
- Network address book
- Keys
 - Generate a new key pair
 - Import an existing key
 - Create a key list
 - Create a threshold key
 - Generate a mnemonic phrase
 - Recover keys from a mnemonic phrase
- Hbars
- Specialized Types
- Pseudorandom Number Generator
- Transactions
 - Transaction ID
 - Modify transaction fields
 - Create an unsigned transaction
 - Manually sign a transaction
 - Submit a transaction
 - Sign a multisignature transaction
 - Get a transaction receipt
 - Get a transaction record
- Scheduled Transaction
 - Schedule ID
 - Create a scheduled transaction
 - Sign a scheduled transaction
 - Delete a scheduled transaction
 - Get schedule info
- Network Response Messages
- Schedule FAQ
- Queries
- General Network Response Messages
- Accounts and hbar
 - Create an account
 - Update an Account
 - Transfer cryptocurrency
 - Approve an allowance
 - Delete an allowance
 - Delete an account
 - Get account balance
 - Get account info
- Network Response Messages
- Consensus Service
 - Create a topic
 - Update a topic
 - Submit a message
 - Delete a topic
 - Get topic messages
 - Get topic info
- Token Service
 - Token ID
 - NFT ID
 - Token types
 - Create a token
 - Custom token fees
 - Update a token
 - Update token custom fees
 - Transfer tokens
 - Delete a token
 - Mint a token
 - Burn a token
 - Freeze an account
 - Unfreeze an account
 - Enable KYC account flag
 - Disable KYC account flag

- Associate tokens to an account
- Dissociate tokens from an account
- Pause a token
- Unpause a token
- Wipe a token
- Atomic swaps
- Get account token balance
- Get token info
- Get NFT info
- Network Response Messages
- File Service
 - Create a file
 - Append to a file
 - Update a file
 - Delete a file
 - Get file contents
 - Get file info
 - Network Response Messages
- Support & Community
 - Hello Future Hackathon
 - Glossary
 - Discord
 - GitHub
 - Stack Overflow
 - Hedera Blog
 - Bug Bounty
 - Hedera Help
 - Documentation Survey
 - Meetups
 - Contributing & Style Guide
 - Brand Guidelines
 - Status Page

keys-and-signatures.md:

Keys and Signatures

Key Types: ECDSA vs Ed25519

A key can be a public key of a supported system Ed25519, ECDSA secp256k1, or an ID of a smart contract. The corresponding algorithm generates public and private keys which are unique to one another. The public key can be shared and visible to other network users in a Network Explorer or REST APIs. The private key is kept secret from the owner and grants access to the owner to modify entities (accounts, tokens, etc.).

Private keys can only be recovered once lost if created with an associated recovery phrase that you can access. Keys are mutable and can be updated once set for an entity. Generally, you will need the current key to sign the transaction to update the keys.

Choosing between ECDSA and Ed25519 Keys

The choice between ECDSA and ED25519 keys primarily depends on your specific use case:

	ECDSA	Ed25519
	Use Cases	Ideal if you want to use MetaMask or need support for more EVM developer tooling. Suited for apps interfacing with Ethereum or EVM-compatible networks due to the associated EVM

address.</td><td>Preferred if key length, security, and performance are important. ECDSA public keys are twice as long for the same level of security.</td></tr></tbody></table>

{% hint style="info" %}

Note: Hedera wallets such as HashPack support both key types.

{% endhint %}

Key Structures

Hedera supports the following key structure types:

		Example	Simple
Account	A single key on an account.	Account Key	{ Key 1 }
Key List	Only one key is required to sign for the account.	Key List	All keys in the key list are required to sign transactions involving the account.
Account Key	Account Key	KeyList	Key 1 Key 2 Key 3
Threshold Key	All three keys in the list are required to sign for the account.	Threshold Key	A subset of keys defined as the threshold are required to sign the transaction that involve the account out of the total number of keys.
Account Key	Account Key	ThresholdKey	Key 1 Key 2 Key 3
Threshold Key	One out of the three keys in the key list is required to sign for the account.		

{% hint style="info" %}

🔔 Key structures can be nested. This means you can have a more complex key system with key lists inside of threshold keys, threshold keys inside keys lists, etc. An example of a nested key list can be viewed [here](#).

{% endhint %}

All transaction types support the above key structures that specify a key field. For a transaction to be successful, the provided signatures must match the defined key structure requirements.

FAQ

<details>

<summary>What is a key in Hedera?</summary>

A key in Hedera can be a public key of a supported system such as ED25519, ECDSA secp256k1, or an ID of a smart contract. The corresponding algorithm generates public and private keys which are unique to one another. The public key can be shared and visible to other network users in a Network Explorer or REST APIs. The private key is kept secret and grants access to the owner to modify entities (accounts, tokens, etc.).

</details>

<details>

<summary>What happens if I lose my private key?</summary>

Private keys can only be recovered once lost if created with an associated recovery phrase that you can access. It's crucial to keep your private keys safe and secure as they grant access to modify your Hedera entities, like accounts and tokens.

</details>

README.md:

cover: ../.gitbook/assets/HH-Eco-Cat-Hero-Desktop-R1 (2).webp

coverY: -37

Core Concepts

schedule-transaction.md:

Schedule Transaction

Overview

A schedule transaction is a transaction that can schedule any Hedera transaction with the ability to collect the required signatures on a Hedera network in preparation for its execution. Unlike other Hedera transactions, this allows you to queue a transaction for execution in the event you do not have all required signatures for the network to immediately process the transaction. This feature is ideal for transactions that require multiple signatures.

When a user creates a schedule transaction, the network creates a schedule entity. The schedule entity receives an entity ID just like accounts, tokens, etc called a schedule ID. The schedule ID is used to reference the schedule transaction that was created. The transaction that is being scheduled is referenced by a scheduled transaction ID. The schedule transaction can be referred to as the outer transaction while the scheduled transaction can be referenced as the inner transaction.

Signatures are appended to the schedule transaction by submitting a ScheduleSign transaction. The ScheduleSign transaction requires the schedule ID of the schedule transaction the signatures will be appended to. In its current design, a schedule transaction has 30 minutes to collect all required signatures before the schedule transaction can be executed or will be deleted from the network. You can delete a schedule transaction by setting an admin key to delete a schedule transaction before it is executed or deleted by the network.

You can request the current state of the a schedule transaction by querying the network for ScheduleGetInfo. The request will return the following information:

Schedule ID

Account ID that created the schedule transaction

Account ID that paid for the creation of the schedule transaction

Transaction body of the inner transaction

Transaction ID of the inner transaction

Current list of signatures

Admin key \ (if any\)

Expiration time

The timestamp of when the transaction was deleted, if true

The design document for this feature can be referenced [here](#).

Schedule Transaction ID

Hedera Transaction IDs are composed of the account ID submitting the transaction and the transaction valid start time in seconds.nanoseconds \ (0.0.1234@126534.126456\). The transaction ID for a schedule transaction will

include "?schedule" at the end of the transaction ID which identifies the transaction as a schedule transaction i.e. 0.0.1.2.3.4@1615422161.673238162?scheduled. The transaction ID of the scheduled \(\inner\) transaction inherits the transaction valid start time and account ID from the schedule \(\outer\) transaction.

Schedule Transaction Receipts

The transaction receipt for a schedule that was created contains the new schedule entity ID and the scheduled transaction ID. The scheduled transaction ID is used to request records for the inner transaction upon successful execution.

Schedule Transaction Records

Transaction records are created when the schedule transaction is created, for each signature that was appended, when the scheduled transaction is executed, and if the schedule transaction was deleted by a user. The record of a schedule transaction includes a schedule reference property which is the ID of the schedule the record is associated with. To get the transaction record for the inner transaction after successful execution, you can do the following:

1. Poll the network for the specified scheduled transaction ID. Once the schedule transaction executes the scheduled transaction successfully, request the record for the scheduled transaction using the scheduled transaction ID.
2. Query a Hedera mirror node for the scheduled transaction ID
3. Run your own mirror node and query for the scheduled transaction ID

scheduled-transaction.md:

Schedule Transaction

Overview

A schedule transaction is a transaction with the ability to collect the required signatures on a Hedera network in preparation for its execution. Unlike other Hedera transactions, this allows you to queue a transaction for execution in the event you do not have all the required signatures for the network to immediately process the transaction. A scheduled transaction is used to create a scheduled transaction. This feature is ideal for transactions that require multiple signatures.

When a user creates a scheduled transaction, the network creates a scheduled entity. The scheduled entity receives an entity ID just like accounts, tokens, etc called a schedule ID. The schedule ID is used to reference the scheduled transaction that was created. The transaction that is being scheduled is referenced by a scheduled transaction ID.

Signatures are appended to the scheduled transaction by submitting a ScheduleSign transaction. The ScheduleSign transaction requires the schedule ID of the scheduled transaction the signatures will be appended to. In its current design, a scheduled transaction has 30 minutes to collect all required signatures before the scheduled transaction can be executed or will be deleted from the network. You can delete a scheduled transaction by setting an admin key to delete a scheduled transaction before it is executed or deleted by the network.

You can request the current state of a scheduled transaction by querying the network for ScheduleGetInfo. The request will return the following information:

Schedule ID
Account ID that created the scheduled transaction
Account ID that paid for the creation of the scheduled transaction
Transaction body of the inner transaction
Transaction ID of the inner transaction
Current list of signatures
Admin key (if any)
Expiration time
The timestamp of when the transaction was deleted, if true

The design document for this feature can be referenced [here](#).

Schedule Transaction ID

Hedera Transaction IDs are composed of the account ID submitting the transaction and the transaction valid start time in seconds.nanoseconds (0.0.1234@1615422161.673238162). The transaction ID for a scheduled transaction will include "?schedule" at the end of the transaction ID which identifies the transaction as a scheduled transaction i.e. 0.0.1234@1615422161.673238162?scheduled. The transaction ID of the scheduled (inner) transaction inherits the transaction valid start time and account ID from the scheduled (outer) transaction.

Schedule Transaction Receipts

The transaction receipt for a schedule that was created contains the new schedule entity ID and the scheduled transaction ID. The scheduled transaction ID is used to request records for the inner transaction upon successful execution.

Schedule Transaction Records

Transaction records are created when the scheduled transaction is created, for each signature that was appended, when the scheduled transaction is executed, and if the scheduled transaction was deleted by a user. The record of a scheduled transaction includes a schedule reference property which is the ID of the schedule the record is associated with. To get the transaction record for the inner transaction after successful execution, you can do the following:

1. Poll the network for the specified scheduled transaction ID. Once the scheduled transaction executes the scheduled transaction successfully, request the record for the scheduled transaction using the scheduled transaction ID.
2. Query a Hedera mirror node for the scheduled transaction ID.
3. Run your own mirror node and query for the scheduled transaction ID.

FAQ

<details>

<summary>What is the difference between a schedule transaction and scheduled transaction?</summary>

A schedule transaction is a transaction that can schedule any Hedera transaction with the ability to collect the required signatures on the Hedera network in preparation for its execution.

A scheduled transaction is a transaction that has already been scheduled.

</details>

<details>

<summary>Is there an entity ID assigned to a schedule transaction?</summary>

Yes, the entity ID is referred to as the schedule ID which is returned in the receipt of the ScheduleCreate transaction.

</details>

<details>

<summary>What transactions can be scheduled?</summary>

In its early iteration, a small subset of transactions will be schedulable. You check out this page for a list of transaction types that are supported today. All other transaction types will be available to schedule in future releases. The complete list of transactions that users can schedule in the future can be found here.

</details>

<details>

<summary>How can I find a schedule transaction that requires my signature?</summary>

The creator of the scheduled transaction can provide you a schedule ID which you specify in the ScheduleSign transaction to submit your signature.

<!-->

You can query a mirror node to return all schedule transactions that have your public key associated with it. This option is not available today, but is planned for the future.

</details>

<details>

<summary>What happens if the scheduled transaction does not have sufficient balance?</summary>

If the scheduled transaction (inner transaction) fee payer does not have sufficient balance then the inner transaction will fail while the schedule transaction (outer transaction) will be successful.

</details>

<details>

<summary>Can you delay a transaction once it has been scheduled?</summary>

No, you cannot delay or modify a scheduled transaction once it's been submitted to a network. You would need to delete the schedule transaction and create a new one with the modifications.

</details>

<details>

<summary>What happens if multiple users create the same schedule transaction?</summary>

The first transaction to reach consensus will create the schedule transaction and provide the schedule entity ID

The other users will get the schedule ID in the receipt of the transaction that was submitted. The receipt status will result in IDENTICALSCHEDULEALREADYCREATED. These users would need to submit a ScheduleSign

transaction to append their signatures to the schedule transaction.

</details>

<details>

<summary>When does the scheduled transaction execute?</summary>

The scheduled transaction executes when the last signature is received.

</details>

<details>

<summary>How often does the network check to see if the scheduled transaction (inner transaction) has met the signature requirement?</summary>

Every time the schedule transaction is signed.

</details>

<details>

<summary>How do you get information about a schedule transaction?</summary>

You can submit a schedule info query request to the network.

</details>

<details>

<summary>When does a scheduled transaction expire?</summary>

A scheduled transaction expires in 30 minutes. In future implementations, we will allow the user to set the time at which the scheduled transaction should execute at, and the transaction will expire at that time.

</details>

<details>

<summary>What does a schedule transaction receipt contain?</summary>

The transaction receipt for a schedule that was created contains the new schedule entity ID and the scheduled transaction ID.

</details>

state-and-history.md:

State and History

Understanding State Machines

A "state machine" represents a conceptual approach to how a program operates: it maintains a "state" and modifies this state in response to specific "transactions." In a "replicated state machine," the duty and accountability for managing this evolving state are distributed across several computers, offering fault tolerance.

Hedera enables a replicated state machine. Numerous nodes, even potentially opposing ones, can consistently maintain the state of a dataset. For example, the HBAR quantity across a group of accounts. As detailed earlier, transactions

are submitted to the network, and subsequently, the hashgraph algorithm assigns them a consensus timestamp and a position in the consensus sequence. Once all nodes reach an agreement on the transaction sequence, they sequentially apply them to the state. This procedure ensures each node's state copy remains consistent. Every node applies (for example, adjusts the payer & recipient balances for an HBAR payment) the transactions to the state following the mutually agreed sequence, thus preserving a uniform state with other nodes at any specific moment.

<figure><figcaption></figcaption></figure>

State vs. History

The latest state (e.g., the HBAR balances of each account) and the history of the transactions that altered that state are two distinct data structures with different properties. State is mutable by definition, constantly changing as transactions are applied to it. In contrast, the history of transactions is generally considered immutable and irreversible. State and history present very different storage burdens. At the high throughput that Hedera can support, history will grow very quickly, increasing the burden of storing it. State will also grow as new accounts, files, and smart contracts are created, but at a slower pace.

Roles of Distributed Technology (DLT) Node

There are three mostly independent functions that a distributed ledger technology (DLT) node can perform:

- Contribute to consensus
- Persist history of transactions
- Persist state

As nodes have limited resources, it is generally the case that a node cannot optimally perform all roles – and choices must be made as to which functions to prioritize.

Priorities of Hedera Mainnet Nodes

For Hedera Mainnet nodes, the priorities contribute to consensus and persists state. The hashgraph, which contains all the transactions that change the state, is constantly pruned after transactions are assigned a place in consensus order. Mainnet nodes can delete older portions of the hashgraph because the algorithm delivers finality – once a transaction has been assigned a timestamp, ordered, and then applied to the state, there is no chance of reversal. Consequently, there is no need to keep historical transactions around in case they might be necessary to apply them in a different order. To prevent such historical transactions from filling up the node's storage, mainnet nodes delete historical transactions.

But there is value in the history being persisted, even if not by mainnet nodes. An auditor might want to determine the identities of the parties that sent HBAR to a given account or the times of those transfers, neither of which would be available from the state (e.g., the balances of the accounts) alone.

Roles of Mirror Nodes

Mirror nodes in the Hedera architecture, in addition to maintaining state, can also store transaction history. A particular mirror can choose whether to store all history, no history, or possibly only a fraction of the history, perhaps only for particular transaction types, particular accounts, etc. In addition to the history, mirror nodes store information that allows them to prove that their history is correct, even for some kinds of partial histories. This prevents a malicious mirror node from lying about what it is storing. A client seeking a

transaction from the past would query an appropriate mirror for the record of that transaction. As the burden of storing history is borne by mirrors and not mainnet nodes, the latter can be optimized for the more fundamental role of consensus and state storage.

FAQ

<details>

<summary>What is the concept of `state` in the Hedera Network?</summary>

The state in the Hedera Network is the current status of all data, like the amount of HBAR in a set of accounts. It is maintained across multiple nodes in a consistent representation, providing fault tolerance. The state constantly changes as transactions are applied to it.

</details>

<details>

<summary>How does Hedera handle history?</summary>

The history of transactions is maintained as a separate data structure from the state. It provides a record of transactions that have changed the state over time. It is usually envisaged as immutable and irreversible. Mirror nodes in the Hedera architecture store the transaction history, while mainnet nodes focus on consensus and state storage.

</details>

<details>

<summary>What are the roles of mainnet nodes and mirror nodes?</summary>

Mainnet nodes prioritize contributing to consensus and persisting state. They delete historical transactions after they are assigned a place in the consensus order. Mirror nodes, on the other hand, store the transaction history and maintain state, providing a record of past transactions for audit purposes.

</details>

transactions-and-queries.md:

description: An overview of Hedera API transactions and queries

Transactions and Queries

Transactions

Transactions are requests sent by a client to a node with the expectation that they are submitted to the network for processing into consensus order and subsequent application to state. Each transaction (e.g. `TokenCreateTransaction()`) has an associated transaction fee compensating the Hedera network for processing and subsequent maintenance in a consensus state. Users can set a maximum transaction fee, which is the amount they are willing to spend. The user is only charged the actual transaction fee.

Transaction ID

Each transaction has a unique transaction ID. The transaction ID is used for the

following:

- Obtaining receipts, records

- Internally by the network for detecting when duplicate transactions are submitted

The transaction ID is composed by using the transaction's valid start time and the account ID of the account that is paying for the transaction. The transaction's valid start time is the time the transaction begins to be processed on the network. The transaction's valid start time can be set to a future date/time. A transaction ID looks something like 0.0.9401@1598924675.82525000 where 0.0.9401 is the transaction fee payer account ID and 1598924675.82525000 is the timestamp in seconds.nanoseconds.

Transactions have a valid duration of up to 180 seconds and begin at the transaction's valid start time. This means that the transaction has up to 180 seconds to be accepted by one of the nodes in the network. If the transaction is not accepted in this timeframe, the transaction will expire. The transaction will have to be created, signed, and submitted again.

A transaction generally includes the following:

- Node Account: the account of the node the transaction is being sent to (e.g. 0.0.3)

- Transaction ID: the identifier for a transaction has two components: the account ID of the paying account plus the transaction's valid start time

- Transaction Fee: the maximum fee the paying account is willing to pay for the transaction

- Valid Duration: the number of seconds that the client wishes the transaction to be deemed valid for, starting at the transaction's valid start time

- Memo: a string of text up to 100 bytes of data (optional)

- Transaction: type of request, for instance, an HBAR transfer or a smart contract call

- Signatures: at minimum, the paying account will sign the transaction as authorization. Other signatures may be present as well.

The lifecycle of a transaction in the Hedera ecosystem begins when a client creates a transaction. Once the transaction is created it is cryptographically signed at a minimum by the account paying for the fees associated with the transaction. Additional signatures may be required depending on the properties set for the account, topic, or token. The client can stipulate the maximum fee it is willing to pay for the processing of the transaction and, for a smart contract operation, the maximum amount of gas. Once the required signatures are applied to the transaction the client then submits the transaction to any node on the Hedera network.

The receiving node validates (for instance, confirms the paying account has sufficient balance to pay the fee) the transaction and, if validation is successful, submits the transaction to the Hedera network for consensus by adding the transaction to an event and gossiping that event to another node. Quickly, that event flows out to all the other nodes. The network receives this transaction exponentially fast via the gossip about gossip protocol. The consensus timestamp for an event (and so the transactions within) is calculated by each node independently calculating the median of the times that the network nodes received that event. You may find more information on how the consensus timestamp is calculated [here](#). The hashgraph algorithm delivers the finality of consensus. Once assigned a consensus timestamp the transaction is then applied to the consensus state in the order determined by each transaction's consensus timestamp. At that point, the transaction fees are also processed. In this manner, every node in the network maintains a consensus state because they all apply the same transactions in the same order. Each node also creates and temporarily stores receipts/records in support of the client, subsequently querying for the status of a transaction.

Nested Transactions

A nested transaction triggers subsequent transactions after executing a top-level transaction. The top-level transaction that a user submits is a parent transaction. For each subsequent transaction, the parent transaction triggers a child transaction as a result of the execution of the parent transaction. An example of a nested transaction is when a user submits the top-level transfer transaction to an account alias that triggers an account creation transaction behind the scenes. This parent/child transaction relationship is also observed with Hedera contracts interacting with HTS precompile. A parent transaction supports up to 999 child transactions since the platform reserves 1000 nanoseconds per user-submitted transaction.

Transaction IDs

Parent and child transactions share the payer account ID and transaction valid start timestamp. The child transaction IDs have an additional nonce value representing the order in which the child transactions were executed. The parent transaction has a nonce value of 0. The nonce value of child transactions increments by 1 for each child transaction executed due to the parent transaction.

Parent Transaction ID: `<mark style="color:red;">payerAccountId</mark>@<mark style="color:blue;">transactionValidStart</mark>`

Child Transaction ID: `<mark style="color:red;">payerAccountId</mark>@<mark style="color:blue;">transactionValidStart</mark>/<mark style="color:green;">nonce</mark>`

Example:

Parent Transaction ID: `<mark style="color:red;">0.0.2252</mark>@<mark style="color:blue;">1640119571.329880313</mark>`
Child 1 Transaction ID: `<mark style="color:red;">0.0.2252</mark>@<mark style="color:blue;">1640119571.329880313</mark>/<mark style="color:green;">1</mark>`
Child 2 Transaction ID: `<mark style="color:red;">0.0.2252</mark>@<mark style="color:blue;">1640119571.329880313</mark>/<mark style="color:green;">2</mark>`

Transaction Records

Nested transaction records are returned by requesting the record for the parent transaction and setting the `setIncludeChildren(<value>)` to true. This returns records for all child transactions associated with the parent transaction. Child transaction records include the parent consensus timestamp and the child transaction ID.

The parent consensus timestamp field in a child transaction record is not populated when the child transaction was triggered before the parent transaction. An example of this case is creating an account using an account alias. The user submits the transfer transaction to create and fund the new account using the account alias. The transfer transaction (parent) triggers the account create transaction (child). However, the child transaction occurs before the parent transaction, so the new account is created before completing the transfer. The parent consensus timestamp is not populated in this case.

Transaction Receipts

Nested transaction receipts can be returned by requesting the parent transaction receipt and setting the boolean value equal to true to return all child transaction receipts.

Child Transaction Fees

The transaction fee for the child transaction is included in the record of the parent transaction. The transaction fee will return zero in the child transaction.

Queries

Queries are processed only by the single node to which they are sent. Clients send queries to retrieve some aspect of the current consensus state, like an account balance. Certain queries are free, but generally, they are subject to fees. The full list of queries can be found [here](#).

A query includes a header that includes a normal HBAR transfer transaction that will serve as how the client pays the node the appropriate fee. There is no way to give partial payment to a node for processing the query, meaning if a user overpaid for the query, the user will not receive a refund. The node processing the query will submit that payment transaction to the network for processing into a consensus statement to receive its fee.

A client can determine the appropriate fee for a query by asking a node for the cost, not the actual data. Such a `COST\ANSWER` query is free to the client.

For more information about query fees, please visit [Hedera API fees overview](#).

Recall:

```
{% hint style="info" %}
Recall
```

Hedera does not have miners or a special group of nodes responsible for adding transactions to the ledger like alternative distributed ledger solutions. Each node's influence on determining the consensus timestamp for an event is proportional to its stake in HBAR.

```
{% endhint %}
```


Once a transaction has been submitted to the network, clients may seek confirmation that it was successfully processed. Multiple confirmation methods are available, varying in the level of information provided, the duration for which the confirmation is available, the degree of trust, and the corresponding cost.

Confirmations

Receipts: Receipts provide minimal information - simply whether or not the transaction was successfully processed into a consensus state. Receipts are generated by default and are persisted for 3 minutes. Receipts are free.

Records: Records provide greater detail about the transaction than do receipts—such as the consensus timestamp it received or the results of a smart contract function call. Records are generated by default but are persisted for 3 minutes.

State proofs (coming soon): When querying for a record, a client can optionally indicate that it desires the network to return a state proof in addition to the record. A state-proof documents network consensus on the contents of that record in the consensus state—this collective assertion includes signatures of most of the network nodes. Because state proofs are cryptographically signed by a supermajority of the network, they are secure and potentially admissible in a court of law.

```
{% content-ref url="../sdks-and-apis/rest-api.md" %}
rest-api.md
{% endcontent-ref %}
```


For a more detailed review of the confirmation methods, please check out this [blog post](#).

FAQ

<details>

<summary>What are the transaction and query fees associated with using Hedera?
</summary>

You can refer to the fees page on Hedera's website for a detailed breakdown of transaction and query costs. If you're looking for an estimation tool, you can use the Hedera fee estimator.

</details>

<details>

<summary>What are transactions?</summary>

Transactions are requests sent by a client to a node with the expectation that they are submitted to the network for processing into consensus order and subsequent application to state. Each transaction has a unique transaction ID composed of the transaction's valid start time and the account ID of the account that is paying for the transaction. This ID is used for obtaining receipts, records, and state proofs and for detecting when duplicate transactions are submitted.

</details>

<details>

<summary>What are queries?</summary>

Queries are requests processed only by the single node to which they are sent. Clients send queries to retrieve some aspect of the current consensus state, like the balance of an account. Certain queries are free, but generally, queries are subject to fees.

</details>

<details>

<summary>What is the difference between receipts and records?</summary>

Receipts provide minimal information - whether or not the transaction was successfully processed into a consensus state. Records provide greater detail about the transaction than receipts, such as the consensus timestamp it received or the results of a smart contract function call.

</details>

account-creation.md:

Account Creation

New accounts are created on the Hedera ledger by submitting a transaction to the network and paying the transaction fee to create the account. The transaction fee to create the account includes the costs required to use network resources, reach consensus amongst the nodes, and share the data across the network.

You will need access to an existing account with enough HBAR to cover the

transaction fee to create the account. Suppose you don't have access to an existing account. In that case, you can use a supported wallet or visit the Hedera Developer Portal to create an account. You can also ask a friend with an existing Hedera account to generously create one for you. Applications can check out the "Auto Account Creation" feature to make free Hedera user accounts.

When an account is created, it is stored in the state on the Hedera network. The current state can be queried from the ledger and viewed in a Network Explorer. Each account has at least one public and private key pair. The private key(s) on the account is used to sign and authorize transactions that involve the account. To view the properties that can be set for an account, check out the "Account Properties" section.

An account can be created through any of the following methods. To create accounts using the SDKs, you will need access to an existing account to pay for the transaction fee to create a new account.

```
{% hint style="warning" %}
:large\orange\diamond: Supported wallets may or may not support creating testnet
and previewnet accounts.
{% endhint %}
```

Hedera Developer Portal	✖ mainnet ✅ testnet ✅ previewnet	https://portal.hedera.com/
Wallets	✅ mainnet 💎 testnet 💎 previewnet	mainnet-access.md
SDKs	✅ mainnet ✅ testnet ✅ previewnet	cryptocurrency

account-properties.md:

Account Properties

Account ID

The account ID is the ID of the account entity on the Hedera network. The account ID includes the shard number, realm number, and an account <shardNum>.<realmNum>.<account>. The account ID is used to specify the account in all Hedera transactions and queries. There can be more than one account ID that can represent an account.

<details>

<summary>Shard Number</summary>

Format: shardNum.realmNum.account

The shard number is the number of the shard the account exists in. A shard is a partition of the data received by the nodes participating in a given shard. Today, Hedera operates in only one shard. This value will remain zero until Hedera operates in more than one shard. This value is non-negative and is 8 bytes.

\

Default: 0

</details>

<details>

<summary>Realm Number</summary>

Format: shardNum.realmNum.account\

\

The realm number is the number of the realm the account exists within a given shard. Today, Hedera operates in only one realm. This value will remain zero until Hedera operates in more than one shard. This value is non-negative and is 8 bytes. The account can only belong to precisely one realm. The realm ID can be reused in other shards.

\

Default: 0

</details>

<details>

<summary>Account</summary>

Format: shardNum.realmNum.account

The account can be one of the following:

\

:black\circle: Account Number \

:black\circle: Account Alias

</details>

Account Number

Each Hedera account has a system-provided account number when the account is created. An account number is a non-negative number of 8 bytes. You can use the account number to specify the account in all Hedera transactions and query requests. Account numbers are unique and immutable. The account number for a newly created account is returned in the transaction receipt or transaction record for the transaction ID that created the account. The account number ID has the following format <mark style="color:purple;"><shardNum>.<realmNum>.<accountNum></mark><mark style="color:blue;">.</mark>

Account Number ID	Description
<code>0.0.10</code>	The account number 10 in account number ID format.

Account Number Alias

All accounts can have an account number alias. An account number alias is a hex-encoded form of the account number prefixed with 20 bytes of zeros. It is an EVM-compatible address that references the Hedera account. The account number alias does not contain the shard ID and realm ID.

This account property is not stored in consensus node state. You will not see

An EVM address account alias is the rightmost 20 bytes of the 32-byte Keccak-256 hash of the ECDSA public key of the account. This calculation is in the manner described by the Ethereum Yellow Paper. Note that the recovery id is not formally part of the public key and is not included in the hash. This is calculated on the consensus nodes using the ECDSA key provided in the auto account creation flow. The EVM address is also commonly known as the public address. The EVM address account ID format is `<mark style="color:purple;"><shardNum>.<realmNum>.<alias></mark>` where `<mark style="color:purple;">alias</mark>` is the EVM address.

The EVM address account and the account number alias are 20-byte values. They can be differentiated because the account number alias is always prefixed with 12 bytes. The EVM address account alias is commonly used in wallets and tools to represent account addresses.

<details>

<summary>EVM Address Account Alias Account ID Example</summary>

The shard number and realm number are set to 0 followed by the EVM address.

```
\
Example\
\
EVM Address: b794f5ea0ba39494ce839613ffffba74279579268\
\
HEX Encoded EVM Address: 0xb794f5ea0ba39494ce839613ffffba74279579268\
\
EVM Address Account Alias Account ID: \
\
0.0.b794f5ea0ba39494ce839613ffffba74279579268
```

</details>

Reference Hedera Improvement Proposal: HIP-583

Account Memo

A memo is like a short note that lives with the account object in the ledger state and can be viewed on a network explorer when looking up the account. This account memo is limited to 100 characters. The account memo is mutable and can be updated or removed from the account at any time. The account key is required to sign the transaction to facilitate any changes to this property.

```
{% hint style="warning" %}
Do not post any private information in the account memo field. This field is
visible to all participants in the network.
{% endhint %}
```

Account Nonce

Accounts on Hedera can submit EthereumTransaction types processed by the Ethereum Virtual Machine (EVM) on a consensus node. The nonce on the account represents a sequentially incrementing count of the transactions submitted by an account through the EthereumTransaction type. The default account nonce value is set to zero.

Reference Hedera Improvement Proposal: HIP-410

Automatic Token Associations

Hedera accounts must generally approve custom tokens before transferring them

into the receiving account. The receiving account must sign the transaction that will associate the tokens, allowing the specified tokens to be deposited into their account. The automatic token association feature allows the account to bypass manually associating the custom token before transferring it into the account.

Accounts can automatically approve up to 5,000 tokens without manually preauthorizing each custom token. Suppose an account needs to hold a balance for custom tokens greater than 5,000. In that case, the account must manually approve each additional token using the transaction to associate the tokens. There is no limit on the total number of tokens an account can hold. This property is mutable and can be changed after it is set.

Maximum Auto-Associations

The property `maxAutoAssociations` of Hedera accounts defines the maximum number of automatic associations allowed. If this is 0, then automatic token associations or airdrops are not allowed, and it requires manual association with the token. This is also if the value is lesser or equal to `usedAutoAssociations`. The default value of the property is -1 for the new automatically created accounts in a way that basically allows unlimited number of auto-associations \[NOT ENABLED]. If the value is a positive number, this puts a limit on the number of auto token associations to that value.

Reference Hedera Improvement Proposal: HIP-23, HIP-904

Balances

When a new account is created, you can specify an initial HBAR balance for the account. The initial HBAR balance for the token is deducted from the account that is paying to create the new account. Creating an account with an initial balance is optional.

A Hedera account can hold a balance of HBAR and custom fungible and non-fungible tokens (NFTs). Account balances can be viewed on a Network Explorer and queried from mirror node REST APIs or consensus nodes.

Token Type	Description
Token ID Example	
HBAR	The native Hedera fungible token used to pay for transaction fees and secure the network.
Fungible Token	Custom fungible tokens created on Hedera. The fungible token ID is represented as <code>0.0.tokenNum</code> ex: <code>0.0.100</code>
Non-Fungible Token (NFTs)	Custom non-fungible tokens (NFTs) created on Hedera. NFT ID is represented as <code>0.0.tokenNum-serialNum</code>.ex: <code>0.0.101-1</code>

Keys

Each account is required to have at least one key upon creation. If a key is not supplied at the time of account creation, the network will reject the transaction. The individual(s) that have access to the account's private key(s) have access to authorize the transfer of tokens into or out of the account and are required to sign transactions that modify the account. Modifying the account includes changing any property, like the balance, keys, memo, etc.

Accounts can optionally have more than one key associated with them. These kinds of accounts are multi-signature accounts meaning you will require more than one key to sign the transaction to change a property on an account or debit HBARS. The signing requirements for a multi-signature account depend on the account's key chosen key structure. For support of key structures and key types, follow the link below.

```
{% content-ref url="../../keys-and-signatures.md" %}  
keys-and-signatures.md  
{% endcontent-ref %}
```

```
{% hint style="danger" %}  
Warning: The private key(s) associated with the account is not to be shared with  
anyone as it will allow others to authorize transactions from your account on  
your behalf. Sharing your private key is like sharing your bank account  
password. Please make sure your private keys are stored in a secure wallet.  
{% endhint %}
```

Receiver Signature Required

Accounts can optionally require the account to sign any transactions depositing tokens into the account. This feature is set to false by default. If this feature is set to true, the account will be required to sign all transactions that deposit tokens into the account. This property is mutable and can be updated after the account is created.

Staking

Staking in Hedera is taking an account and associating the HBAR balance to a node in the network. Custom fungible or non-fungible token balances an account holds do not contribute to staking on the network. The purpose of staking accounts to a node on the network is to strengthen the security of the network. To contribute to the security of the network, staked accounts can earn rewards in HBAR. Please see this guide for additional information about the staking rewards program. Contracts can also stake their accounts to earn rewards.

```
{% hint style="info" %}  
An account can only stake to one node or one account at any given time.  
{% endhint %}
```

<details>

<summary>Staked Node ID</summary>

An account can optionally elect to stake its HBAR to a node in the Hedera network. The staked node ID is the node an account can stake to. The full balance of the account is staked to the node. Do not confuse the node ID with the node's account ID. If you stake to the node's account ID, your account will not earn staking rewards. \

\

The staked account balance is liquid at all times. This means you can transfer HBAR tokens in and out of the account, and your account will continue to be staked to the node without disruption. \

\

There is no lock-up period. This means the HBAR tokens in your account are not held for a period of time before you can use them. \

\

The node ID for a node can be found here or can be queried from the nodes REST API.\

\

Example:\

\

Node ID: 1\

</details>

<details>

<summary>Staked Account ID</summary>

An account can optionally elect to stake its HBAR to another account in the Hedera network. This is known as indirect staking. The staked account ID is the ID of the account to stake to. The full balance of the account is staked to the specified account. \

\ There is no lock-up period and the balance is always liquid just like staking to a node. \

\ Accounts that stake to another account do not earn the staking rewards. For example, If account A is staked to account B, account B will need to be staked to a node in order to contribute to network security and earn staking rewards. Account B will earn the rewards for staking when staked to a node for both the HBAR balances in both Account A + Account B. Account A will not earn rewards for staking. \

\ Example:\

\ Account ID: 0.0.10

</details>

<details>

<summary>Decline to Earn Staking Rewards</summary>

Accounts can decline to earn staking rewards when they stake to a node or an account. The staked account still contributes to the staking weight of the node, but does not earn rewards or is calculated as part of the payment of the rewards to the other accounts that have elected to earn rewards. By default, all staked accounts will earn rewards unless this boolean flag is set to true. This election can be changed by updating the account properties. Hedera treasury accounts enable this flag to decline earning staking rewards. \

\ Default: true (all accounts accept earning staking rewards if the account is staked)

</details>

Staking Information

The network stores the staking metadata for an account and contract accounts. This information is returned in account information query requests (AccountInfoQuery or ContractInfoQuery). The staking metadata for an account includes the following information:

decline\reward: whether or not the account declined to earn staking rewards
stake\period\start: The staking period during which either the staking settings for this account or contract changed (such as starting staking or changing staked\node\id) or the most recent reward was earned, whichever is later. If this account or contract is not currently staked to a node, then this field is not set. The stake period is 24 hours, starting UTC midnight.

pending\reward: The amount in tinybars that will be received in the next staking reward payment

staked\to\me: The total tinybar balance of all accounts staked to this account

staked\id: ID of the account or node to which this account or contract is staking

staked\account\id: The account to which this account or contract is staking

to
 staked\node\id: The ID of the node this account or contract is staked to

Reference Hedera Improvement Proposal: HIP-406

Auto Renewals & Expiration

{% hint style="warning" %}
Auto-renewals and expiration (rent) are currently not enabled.
{% endhint %}

Like the other Hedera entities, accounts take up network storage. To cover the cost of storing an account, a renewal fee will be charged for the storage utilized on the network. This feature is not enabled on the network today; however, in the future, when it is enabled, the account must have sufficient funds to pay for the renewal fees.

The amount charged for renewal will be charged every pre-determined period in seconds. The interval of time the account will be charged is the auto-renew period. The system will automatically charge the account renewal fees. If the account does not have an HBAR balance, it will be suspended for one week before it is deleted from the ledger. You can renew an account during the suspension period.

<details>

<summary> Expiration Time</summary>

The effective consensus timestamp at (and after) which the entity is set to expire.

</details>

<details>

<summary>Auto Renew Account</summary>

The auto renew account is the account that will be used to pay for the auto renewal fees. If there is no auto renew account specified, the auto renewal fees will be charged to the account.

</details>

<details>

<summary>Auto Renew Period</summary>

The interval at which this account will be charged the auto renewal fees. The maximum auto renew period for an account is be limited to 3 months (8000001 sec seconds). The minimum auto renew period is 30 days. The auto renew period is mutable and can be updated at any time. If there are insufficient funds, then it extends as long as possible. If it is empty when it expires, then it is deleted.

</details>

Reference Hedera Improvement Proposal: HIP-16

auto-account-creation.md:

Auto Account Creation

Auto account creation is a unique flow in which applications, like wallets and

exchanges, can create free user "accounts" instantly, even without an internet connection. Applications can make these by generating an account alias. The alias account ID format used to specify the account alias in Hedera transactions comprises the shard ID, realm ID, and account alias `<mark style="color:purple;"><shardNum>.<realmNum>.<alias></mark>`. This is an alternative account identifier compared to the standard account number format `<mark style="color:purple;"><shardId>.<realmId>.<accountNum></mark><mark style="color:blue;">.</mark>`

The account alias can be either one of the supported types:

`<details>`

`<summary>Public Key</summary>`

The public key alias can be an ED25519 or ECDSA secp256k1 public key type. \

\

Example\

\

ECDSA secp256k1 Public Key:\

02d588ec1000770949ab77516c77ee729774de1c8fe058cab6d64f1b12ffc8ff07\

\

DER Encoded ECDSA secp256k1 Public Key Alias:\

302d300706052b8104000a03220002d588ec1000770949ab77516c77ee729774de1c8fe058cab6d64f1b12ffc8ff07\

\

ECDSA secp256k1 Public Key Alias Account ID: \

0.0.302d300706052b8104000a03220002d588ec1000770949ab77516c77ee729774de1c8fe058cab6d64f1b12ffc8ff07

\

\

\

\

EDDSA ED25519 Public Key:\

1a5a62bb9f35990d3fea1a5bb7ef6f1df0a297697adef1e04510c9d4ecc5db3f\

\

DER Encoded EDDSA ED25519 Public Key Alias:\

302a300506032b65700321001a5a62bb9f35990d3fea1a5bb7ef6f1df0a297697adef1e04510c9d4ecc5db3f\

\

EDDSA ED25519 Public Key Alias Account ID: \

0.0.302a300506032b65700321001a5a62bb9f35990d3fea1a5bb7ef6f1df0a297697adef1e04510c9d4ecc5db3f

`</details>`

`<details>`

`<summary>EVM Address</summary>`

The EVM address alias is created by using the rightmost 20 bytes of the 32 byte Keccak-256 hash of an ECDSA secp256k1 public key. This calculation is in the manner described by the Ethereum Yellow Paper. The EVM address is not equivalent to the ECDSA public key. \

\

The acceptable format for Hedera transactions is the EVM Address Alias Account ID. The acceptable format for Ethereum public addresses to denote an account address is the hex encoded public address. \

\

Example\

\

EVM Address: b794f5ea0ba39494ce839613ffffba74279579268\

\

HEX Encoded EVM Address: 0xb794f5ea0ba39494ce839613ffffba74279579268\

\
EVM Address Alias Account ID: 0.0.b794f5ea0ba39494ce839613ffffba74279579268

</details>

The <mark style="color:purple;"><shardNum>.<realmNum>.<alias></mark> format is only acceptable when specified in the TransferTransaction, AccountInfoQuery, and AccountBalanceQuery transaction types. If this format is used to specify an account in any other transaction type, the transaction will not succeed.

Reference Hedera Improvement Proposal: HIP-583

Auto Account Creation Flow

1. Create an account alias

Create an account alias and convert it to the alias account ID format. The alias account ID format requires appending the shard number and realm numbers to the account alias. This form of account is purely a local account, i.e., not registered with the Hedera network.

2. Deposit tokens to the account alias account ID

Once the alias account ID is created, applications can create a transaction to transfer tokens to the alias account ID for users. Users can transfer HBAR, custom fungible or non-fungible tokens to the alias account ID. This will trigger the creation of the official Hedera account. When using the auto account creation flow, the first token transferred to the alias account ID is automatically associated to the account.

The initial transfer of tokens to the alias account ID will do a few things:

1. The system will first create a system-initiated transaction to create a new account on Hedera. This is to create a new account on Hedera officially. This transaction occurs one nanosecond before the subsequent token transfer transaction.
2. After the new account is created, the system will assign a new account number, and the account alias will be stored with the account in the alias field in the state. The new account will have an "auto-created account" set in the account memo field.
For accounts created using the public key alias, the account will be assigned the account public key that matches the alias public key.
For an account created via an EVM address alias, the account will not have an account public key, creating a hollow account.
3. Once the new account is officially created, the token transfer transaction instantiated by the user will transfer the tokens to the new account.
4. The account specified to pay for the token transfer transaction fees will also be charged the account creation transaction fees in tinybar.

The above interactions introduce the concept of parent and child transactions. The parent transaction is the transaction that represents the transfer of tokens from the sender account to the destination account. The child transaction is the transaction the system initiated to create the account. This concept is important since the parent transaction record or receipt will not return the new account number ID. You must get the transaction record or receipt of the child transaction. The parent and child transactions will share the same transaction ID, except the child transaction has an added nonce value.

{% hint style="info" %}

parent transaction: the transaction responsible for transferring the tokens to the alias account ID destination account.

child transaction: the transaction that created the new account

{% endhint %}

3. Get the new account number

You can obtain the new account number in any of the following ways:

Request the parent transaction record or receipt and set the child transaction record boolean flag equal to true.

Request the transaction receipt or record of the account create transaction by using the transaction ID of the parent transfer transaction and incrementing the nonce value from 0 to 1.

Specify the account alias account ID in an AccountInfoQuery transaction request. The response will return the account's account number account ID.

Inspect the parent transfer transaction record transfer list for the account with a transfer equal to the token transfer value.

Auto Account Creation: EVM Address Alias

Accounts created with the auto account creation flow using an EVM address alias result in creating a hollow account. Hollow accounts are incomplete accounts with an account number and alias but not an account key. The hollow account can accept token transfers into the account in this state. It cannot transfer tokens from the account or modify any account properties until the account key has been added and the account is complete.

To update the hollow account into a complete account, the hollow account needs to be specified as a transaction fee payer account for a Hedera transaction. It must also sign the transaction with the ECDSA private key corresponding to the EVM address alias. When the hollow account becomes a complete account, you can use the account to pay for transaction fees or update account properties as you would with regular Hedera accounts.

Examples

<details>

<summary>Auto-create an account using a public key alias</summary>

:black\circle: Java \
:black\circle: JavaScript \
:black\circle: Go

</details>

<details>

<summary>Auto-create an account using an EVM address (public address) alias</summary>

:black\circle: Java \
:black\circle: JavaScript \
:black\circle: Go

</details>

README.md:

Accounts

Accounts are the central starting point when interacting with the Hedera network and using Hedera Services. A Hedera account is an entity, a distinct object type, stored in the ledger, that holds tokens. Accounts can hold the native Hedera fungible token (HBAR), custom fungible, and custom non-fungible tokens (NFTs) created on the Hedera network.

The Hedera native token HBAR (\hbar) is a utility token primarily used to pay for transactions and query fees when interacting with the network. The HBAR symbol is represented as " \hbar ." Applications may reference HBAR as the token denomination; however, the network returns information in tinybars ($t\hbar$), a denomination of HBAR. 100,000,000 $t\hbar$ are equivalent to 1 \hbar . This includes things like transaction fees or accounts HBAR balances.

You interact with the network by submitting transactions that modify the ledger's state or submitting query requests that read data from the ledger. Most transactions and queries have a transaction fee that is charged in HBAR. Unlike custom tokens users create on the Hedera network, no token ID represents the native HBAR token.

Account Creation	account-creation.md
Auto Account Creation	auto-account-creation.md
Account Properties	account-properties.md

FAQ

<details>

<summary>What is a Hedera account?</summary>

A Hedera account is a unique entity in the Hedera Network that can hold tokens. These can be Hedera's native fungible token (HBAR), custom fungible, or non-fungible tokens (NFTs).

</details>

<details>

<summary>How are new accounts created on Hedera?</summary>

New accounts are created by submitting a transaction to the network and paying the transaction fee. You'll need access to an existing account with sufficient HBAR to cover this fee. If you don't have access to an existing account, you can use a supported wallet, visit the Hedera Developer Portal, or use the "Auto Account Creation" feature for applications.

</details>

<details>

<summary>What is the 'Auto Account Creation' feature?</summary>

Auto Account Creation allows applications to generate free user accounts instantly, even without an internet connection, by creating an account alias.

</details>

<details>

<summary>What is a "hollow" account?</summary>

If an account is created with an EVM address alias via Auto Account Creation, it results in a "hollow" account. This account has an account number and alias but

no account key. It can accept token transfers but cannot transfer tokens or modify account properties until the account key has been added, completing the account.

</details>

gossip-about-gossip.md:

Gossip About Gossip

Hashgraph consensus uses a gossip protocol. This means that a member such as Alice will choose another member at random, such as Bob, and then Alice will tell Bob all of the information she knows so far. Alice then repeats with a different random member. Bob repeatedly does the same, and all other members do the same. In this way, if a single member becomes aware of new information, it will spread exponentially fast through the community until every member is aware of It.

The synchronization of information between two members through the gossip protocol is called a gossip sync. Upon completion of a gossip sync, each participating member commemorates the gossip sync with an event. An event is stored in memory as a data structure composed of a timestamp, an array of zero or more transactions, two parent hashes, and a cryptographic signature. The two parent hashes are the hash of the last event created by the self-parent prior to the gossip sync and the hash of the last event created by the other-parent prior to the gossip sync. For example, if Alice and Bob perform a gossip sync, Alice would create a new event commemorating the gossip sync where the self-parent hash would be the hash of the last event Alice created prior to the gossip sync and the other-parent hash would be the hash of the last event Bob created prior to the gossip sync. Bob would also create an event commemorating the gossip sync, but the self-parent hash would be the hash of the last event he created before the gossip sync and the other-parent hash would be the hash of the last event Alice created before the gossip sync. Gossip continues until all members have received the newly created event.

Gossip About Gossip

The history of how these events are related to each other through their parent hashes is called gossip about gossip. This history expresses itself as a type of directed acyclic graph \(\text{DAG}\), a graph of hashes, or a hashgraph. The hashgraph records the history of how members communicated. It grows directionally over time as more gossip syncs take place and events are created. All members keep a local copy of the hashgraph which continues to update as members sync with one another.

These hashgraphs may be slightly different at any given moment, but they will always be consistent. Consistent means that if \([Alice]\) and \([Bob]\) both contain event x , then they will both contain exactly the same set of ancestors for x , and will both contain exactly the same set of edges between those ancestors.

Each event contains the following:

- Timestamp
- Two hashes of two events below itself
 - Self-parent
 - Other-parent
- Transactions
- Digital signature

Item	Description
:---	:---
Timestamp:	The timestamp of when the member created the event commemorating

```
the gossip sync |
| Transactions: | The event can hold zero or more transactions |
| Hash 1: | Self-parent hash |
| Hash 2: | Other-parent hash |
| Digital Signature: | Cryptographically signed by the creator of the event |
```

README.md:

```
---
description: Distributed consensus algorithm
---
```

Hashgraph Consensus Algorithm

The hashgraph consensus algorithm enables distributed consensus in an innovative, efficient way. Hashgraph is a distributed consensus algorithm and data structure that is fast, fair, and secure. This indirectly creates a trusted community, even when members do not necessarily trust each other.

The hashgraph consensus algorithm and platform code are open-source under an Apache 2.0 license.

```
{% embed url="https://www.youtube.com/watch?v=cje1vuVKhwY&t=5s" %}
```

Performance

Cost

The hashgraph is inexpensive, in the sense of avoiding proof-of-work. Individuals and organizations running hashgraph nodes do not need to purchase expensive custom mining rigs. Instead, they can run readily available, cost-effective hardware. The hashgraph is 100% efficient, wasting no resources on computations that slow it down.

Efficiency

The hashgraph is 100% efficient, as that term is used in the blockchain community. In blockchain, work is sometimes wasted mining a block that later is considered stale and is discarded by the community. In hashgraph, the equivalent of a "block" never becomes stale. Hashgraph is also efficient in its use of bandwidth. Whatever is the amount of bandwidth required merely to inform all the nodes of a given transaction (even without achieving consensus on a timestamp for that transaction), hashgraph adds only a very small overhead beyond that absolute minimum. Additionally, hashgraph's voting algorithm does not require any additional messages be sent in order for nodes to vote (or those votes to be counted) beyond those messages by which the community learned of the transaction itself.

Throughput

The hashgraph is fast. It is limited only by the bandwidth. If each member has enough bandwidth to download and upload a given number of transactions per second, the system as a whole can handle close to that many. Even a fast home internet connection could be fast enough to handle all of the transactions of the entire VISA card network, worldwide.

State Efficiency

Once an event occurs, within seconds everyone in the community will know where it should be placed in history with 100% certainty. More importantly, everyone will know that everyone else knows this. At that point, they can just incorporate the effects of the transaction and, unless needed for future audit

or compliance, then discard it. So in a minimal cryptocurrency system, each member would only need to store the current balance of each account that isn't empty. They wouldn't need to remember the full history of the transactions that resulted in those balances all the way back to 'genesis'.

Security

Asynchronous Byzantine Fault Tolerance

The hashgraph consensus algorithm is asynchronous Byzantine Fault Tolerant. This means that no single member (or small group of members) can prevent the community from reaching a consensus. Nor can they change the consensus once it has been reached. Each member will eventually reach a point where they know for sure that they have reached consensus. Blockchain does not have a guarantee of Byzantine agreement, because a member never reaches certainty that agreement has been achieved (there's just a probability that rises over time). Blockchain is also non-Byzantine because it doesn't automatically deal with network partitions. If a group of miners is isolated from the rest of the internet, that can allow multiple chains to grow, which conflict with each other on the order of transactions.

It is worth noting that the term "Byzantine Fault Tolerant" (BFT) is sometimes used in a weaker sense by other consensus algorithms. But here, it is used in its original, stronger sense that (1) every member eventually knows consensus has been reached, (2) attackers may collude, and (3) attackers even control the internet itself (with some limits). Hashgraph is Byzantine, even by this stronger definition.

There are different degrees of BFT, depending on the assumptions made about the network and transmission of messages. The strongest form of BFT is asynchronous BFT- meaning that it can achieve consensus even if malicious actors are able to control the network and delete or slow down messages of their choosing. The only assumptions made are that more than 2/3 are following the protocol correctly and that if messages are repeatedly sent from one node to another over the internet, eventually one will get through, and then eventually another will, and so on. Some systems are partially asynchronous, which are secure only if the attackers do not have too much power and do not manipulate the timing of messages too much. For instance, a partially asynchronous system could prove Byzantine under the assumption that messages get passed over the internet in ten seconds. This assumption ignores the reality of botnets, Distributed Denial of Service attacks, and malicious firewalls.

ACID Compliance

The hashgraph is ACID compliant. ACID (Atomicity, Consistency, Isolation, Durability) is a database term and applies to the hashgraph when it is used as a distributed database. A community of nodes uses it to reach a consensus on the order in which transactions occurred. After reaching consensus, each node feeds those transactions to that node's local copy of the database, sending in each one in the consensus order. If the local database has all the standard properties of a database (ACID), then the community as a whole can be said to have a single, distributed database with those same properties. In blockchain, there is never a moment when you know that consensus has been reached, so it would not be ACID compliant.

Distributed Denial of Service (DDoS) Attack Resilience

One form of Denial of Service (DoS) attack occurs when an attacker is able to flood an honest node on a network with meaningless messages, preventing that node from performing other (valid) duties and roles. A Distributed Denial of Service (DDoS) uses public services or devices to unwittingly amplify that DoS attack - making them an even greater threat.

In a distributed ledger, a DDoS attack could target the nodes that contribute to

the definition of consensus and, potentially, prevent that consensus from being established.

Hashgraph is DDoS resilient as it empowers no single node or a small number of nodes with special rights or responsibilities in establishing consensus. Both Bitcoin and hashgraph are distributed in a way that resists DDoS attacks. An attacker might flood one member or miner with packets, to temporarily disconnect them from the internet. But the community as a whole will continue to operate normally. An attack on the system as a whole would require flooding a large fraction of the members with packets, which is more difficult. There have been a number of proposed alternatives to blockchain-based on leaders or round-robin. These have been proposed to avoid the proof-of-work costs of Bitcoin. But they have the drawback of being sensitive to DDoS attacks. If the attacker attacks the current leader, and switches to attacking the new leader as soon as one is chosen, then the attacker can freeze the entire system while still attacking only one computer at a time. Hashgraph avoids this problem, while still not needing proof-of-work.

Fairness

Hashgraph is fair because there is no leader or miner given special permissions for determining the consensus timestamp assigned to a transaction. Instead, the consensus timestamp for transactions are calculated via a voting process in which the nodes collectively and democratically establish the consensus. We can distinguish between three aspects of fairness.

Fair Access

Hashgraph is fundamentally fair because no individual can stop a transaction from entering the system, or even delay it very much. If one (or few) malicious nodes attempts to prevent a given transaction from being delivered to the rest of the community and so be added into consensus, then the random nature of the gossip protocol will ensure that the transaction flows around that blockage.

Fair Timestamps

Hashgraph gives each transaction a consensus timestamp that reflects when the majority of the network members received that transaction. This consensus timestamp is "fair", because it is not possible for a malicious node to corrupt it and make it differ by very much from that time. Every transaction is assigned a consensus time, which is the median of the times at which each member says it first received it. Received here refers to the time that a given node was first passed the transaction from another node through gossip. This is part of the consensus, and so has all the guarantees of being Byzantine. If more than 2/3 of participating members are honest and have reliable clocks on their computer, then the timestamp itself will be honest and reliable, because it is generated by an honest and reliable member or falls between two times that were generated by honest and reliable members. Because hashgraph takes the median of all these times, the consensus timestamp is robust. Even if a few of the clocks are a bit off, or even if a few of the nodes maliciously give times that are far off, the consensus timestamp is not significantly impacted.

This consensus timestamping is useful for things such as a legal obligation to perform some action by a particular time. There will be a consensus on whether an event happened by a deadline, and the timestamp is resistant to manipulation by an attacker. In a blockchain, each block contains a timestamp, but it reflects only a single clock: the one on the computer of the miner who mined that block.

Fair Transaction Order

Transactions are put into order according to their timestamps. Because the timestamps assigned to individual transactions are fair, so is the resulting order. This is critically important for some use cases. For example, imagine a

stock market, where Alice and Bob both try to buy the last available share of a stock at the same moment for the same price. In a blockchain, a miner might put both of those transactions in a single block, and have complete freedom to choose what order they occur. Or the miner might choose to only include Alice's transaction, and delay Bob's to a future block. In hashgraph, there is no way for an individual to unduly affect the consensus order of those transactions. The best Alice can do is to invest in a better internet connection so that her transaction reaches everyone before Bob's. That's the fair way to compete.

FAQ

<details>

<summary>What is the hashgraph consensus algorithm? How does it work?</summary>

The hashgraph consensus algorithm is a distributed consensus mechanism used by Hedera. It uses a data structure called a hashgraph, and a consensus mechanism called the Gossip protocol. This combination allows for fast, fair, and secure consensus. The algorithm works by each node in the network sharing information (or "gossiping") about the transactions it knows about with other nodes in random order.

</details>

<details>

<summary>How secure is the hashgraph consensus algorithm?</summary>

Hashgraph is secure because it is asynchronous Byzantine Fault Tolerant (aBFT). This means that no single member or small group of members can prevent the community from reaching a consensus or changing the consensus once it has been reached. It is also ACID compliant when used as a distributed database, and it is resilient to Distributed Denial of Service (DDoS) attacks.

</details>

<details>

<summary>What is virtual voting?</summary>

Virtual voting is an integral part of the hashgraph consensus algorithm. It allows nodes to know what others would vote for without needing actual votes sent over the internet. This is accomplished by examining the history of gossip (who spoke to whom and in what order) to determine how a node would vote based on the information it is likely to have.

</details>

virtual-voting.md:

Virtual Voting

It is not enough to ensure that every member knows every event. It is also necessary to agree on a linear ordering of the events, and thus of the transactions recorded inside the events. Most Byzantine fault tolerance protocols without a leader depend on members sending each other votes...Some of these protocols require receipts on votes sent to everyone...And they may require multiple rounds of voting, which further increases the number of voting messages sent.

This pure voting approach becomes bandwidth prohibitive and impractical in a

network of any significant size but has the properties of being the fairest and most secure method of reaching consensus. The hashgraph algorithm implements voting that achieves the same fair and secure properties but is also very fast and practical. It accomplishes this through virtual voting.

The hashgraph algorithm does not require any votes to be sent across the network to calculate the votes of each member. Members can calculate every other member's votes by internally looking at each of their copies of the hashgraph and applying the virtual voting algorithm. Votes are calculated locally as a function of the ancestors of a given event.

This virtual voting has several benefits. In addition to saving bandwidth, it ensures that members always calculate their votes according to the rules. If Alice is honest, she will calculate virtual votes for the virtual Bob that are honest. Even if the real Bob is a cheater, he cannot attack Alice by making the virtual Bob vote incorrectly.

With this virtual voting algorithm, Byzantine agreement is guaranteed.

Virtual voting happens in 3 steps:

1. Divide Rounds
2. Decide Fame
3. Find Order

Divide Rounds

To begin the process of virtual voting, we must first define rounds and witnesses. In the hashgraph history, the first event for a member's node is that node's first witness. The first witness is the beginning of the first round $\setminus(r)$ for that node. All subsequent events are part of that first round until a new witness is discovered. A witness is discovered when a node creates a new event that can strongly see $\frac{2}{3}$ of the witnesses in the current round. For example, event w can strongly see event x when w can trace its ancestry through parent relationships that pass through other events that reside on at least $\frac{2}{3}$ of the member nodes. When an event is determined to strongly see $\frac{2}{3}$ of the witness of the current round, that event is considered the next witness for that node. That new witness is the first event in the next round $\setminus(r+1)$ for that node. Each event is assigned a round as the event is added to the hashgraph.

text

procedure divideRounds

```
for each event x
  r <- max round of parents of x (or 1 if none exist)
  if x can strongly see more than 2n/3 round r witnesses
    x.round <- r+1
  else
    x.round <- r
  x.witness <- (x has no self parent)
               or (x.round > x.selfParent.round)
```

Decide Fame

The next step is deciding whether a witness is a famous witness or not. A witness is famous if many of the witnesses in the next round can see it, and it is not famous if many can't. Event A can see Event B if Event B is an ancestor of Event A. When deciding the fame of Witness A, we must look at the witnesses of the following round. If the witnesses of the following round can see Witness A, they count as a vote in favor of Witness A's fame. Likewise, if a witness in the next round can not see Witness A, then that witness' vote is that Witness A is not famous. In order for Witness A to be considered famous, a future witness must be able to strongly see that at least $\frac{2}{3}$ of voting witnesses have voted in

favor of Witness A being famous. If $\frac{2}{3}$ of voting witnesses have voted that Witness A is not famous, then Witness A will be decided to be not famous.

Find Order

Now that we have calculated the all witnesses of a round to be famous or not famous, we can determine the order of events that occurred before the famous witness events. This is done by calculating:

1. The round received for all events that have yet to be ordered and that have occurred before a round where the fame of all witnesses has been decided. The event's round received is the first round where all famous witnesses of that round can see \ (or are descendants of\) the event in question.
2. The timestamp for each event. This is done by gathering the earliest ancestors of the famous witnesses of the round received that are also descendants of the event in question, and taking the median timestamp of those gathered events.
3. The ordering of events first by: round received, consensus timestamp, then signature.

hedera-mirror-node.md:

Hedera Mirror Node

The Hedera Consensus Service (HCS) is a gRPC API endpoint on the mirror node to stream HCS messages. It offers the ability to subscribe to HCS topics and receive messages for the topic subscribed to. API docs for the mirror nodes can be found here:

```
{% content-ref url="../../../sdks-and-apis/rest-api.md" %}  
rest-api.md  
{% endcontent-ref %}
```

```
{% content-ref url="../../../sdks-and-apis/hedera-consensus-service-api.md" %}  
hedera-consensus-service-api.md  
{% endcontent-ref %}
```

Mainnet

The non-production public mainnet mirror node serves to help developers build their applications without having to run their own mirror node. For production-ready mainnet mirror nodes, please check out Arkhia, Dragonglass, Hgraph, or Ledger Works. When building your Hedera client via SDK, you can use `setMirrorNetwork()` and enter the public mainnet mirror node endpoint. The gRPC API requires TLS. The following SDK versions support TLS:

```
Java: v2.0.6+  
JavaScript: v2.0.23+  
Go: v2.1.9+
```

```
{% hint style="warning" %}  
Public mainnet mirror node requests per second (RPS) are currently throttled at  
50 per IP address. These configurations may change in the future depending on  
performance or security considerations. At this time, no authentication is  
required.  
{% endhint %}
```

```
{% tabs %}  
{% tab title="Java" %}  
java  
//You will need to upgrade to v2.0.6 or higher  
Client client = Client.forMainnet();
```

```
client.setMirrorNetwork(Collections.singletonList("mainnet.mirrornode.hedera.com:443"))
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//You will need to upgrade to v2.0.23 or higher
```

```
const client = Client.forMainnet()
```

```
client.setMirrorNetwork("mainnet.mirrornode.hedera.com:443")
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
client := hedera.ClientForMainnet()
```

```
client.SetMirrorNetwork([]string{"mainnet.mirrornode.hedera.com:443"})
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
{% hint style="info" %}
```

```
Mainnet Mirror Node Endpoint: mainnet.mirrornode.hedera.com:443\
```

```
\
```

```
REST API Mainnet Root Endpoint: https://mainnet.mirrornode.hedera.com
```

```
{% endhint %}
```

Testnet

The endpoints provided below allow developers to access the testnet mirror node, which contains testnet transaction data.

```
{% hint style="info" %}
```

```
HCS Testnet Mirror Node Endpoint: testnet.mirrornode.hedera.com:443
```

```
REST API Testnet Root Endpoint: https://testnet.mirrornode.hedera.com
```

```
{% endhint %}
```

Previewnet

The endpoints provided below allow developers to access the previewnet mirror node, which contains previewnet transaction data.

```
{% hint style="info" %}
```

```
HCS Previewnet Mirror Node Endpoint: previewnet.mirrornode.hedera.com:443
```

```
REST API Preview Testnet Root Endpoint: https://previewnet.mirrornode.hedera.com
```

```
{% endhint %}
```

one-click-mirror-node-deployment.md:

One Click Mirror Node Deployment

Google Cloud Platform Marketplace

Deploy your own mirror node with just a few clicks! The Hedera Mirror Node is open-source software and does not carry an associated license or deployment fee. However, users are responsible for paying for the compute resource used, the data retrieved from Google Cloud Storage buckets, and any other Google Cloud Platform services used during this deployment.

► Before you proceed to the next step, obtain the GCP requester pay information:

<details>

<summary>How To Obtain Google Cloud Platform Requester Pay Information</summary>

In this step, you will generate your Google Cloud Platform HMAC access keys. These keys are needed to authenticate requests between your machine and Google Cloud Storage. They are similar to a username and password. Follow these steps to retrieve your access key, secret, and project ID:

Create a new project and link your billing account.
From the left navigation bar, select Cloud Storage > Settings.
Click the Interoperability tab and scroll down to the User account HMAC section.
If you don't already have a default project set, set it now.
Click create keys to generate access keys for your account.

</details>

Prerequisites

Click [here](#) to get started; you will need the following details you obtained from the previous step:

Importer GCP access key
Importer GCS bucket name
Mainnet: hedera-mainnet-streams
Testnet: hedera-stable-testnet-streams-2024-02
Importer GCP billing project ID
Importer GCP secret key

Once you deploy your mirror node, you can access the mirror node via the gRPC API or the REST API.

gRPC API

Use the following terminal commands:

```
bash
GRPCIP=$(kubectl get service/hedera-mirror-node-1-grpc -n default -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

grpcurl -plaintext "${GRPCIP}:5600" list
```

REST API

For the REST API, use these terminal commands:

```
bash
RESTIP=$(kubectl get service/hedera-mirror-node-1-rest -n default -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

curl -s "http://${RESTIP}/api/v1/transactions?limit=1"
```

Alternatively, you can also submit a get request from your browser:

From your Kubernetes Engine navigation bar, click on "Workloads."
Click on "hedera-mirror-node-1-rest"
Navigate down to "Exposing services" and click on the "Endpoints" link

Example GET request:

```
bash
```

`http://<yourEndpoint>/api/v1/transactions`

```
{% content-ref url="../../../sdks-and-apis/rest-api.md" %}
rest-api.md
{% endcontent-ref %}
```

README.md:

```
---
description: Store history and cost-effectively query data
---
```

Mirror Nodes

Mirror nodes provide a way to store and cost-effectively query historical data from the public ledger while minimizing the use of Hedera network resources. Mirror nodes support the Hedera network services currently available and can be used to retrieve the following information:

- Transactions and records
- Event files
- Balance files

Understanding Mirror Nodes

Hedera Mirror Nodes receive information from Hedera network consensus nodes, either mainnet or testnet, and provide a more effective means to perform:

- Queries
- Analytics
- Audit support
- Monitoring

While mirror nodes receive information from the consensus nodes, they do not contribute to consensus themselves. The trust of Hedera is derived based on the consensus reached by the consensus nodes. That trust is transferred to the mirror nodes using signatures, chain of hashes, and state proofs.

To make the initial deployments easier, the mirror nodes strive to take away the burden of running a full Hedera node through the creation of periodic files that contain processed information (such as account balances or transaction records) and have the full trust of the Hedera consensus nodes. The mirror node software reduces the processing burden by receiving pre-constructed files from the network, validating those, populating a database, and providing REST APIs.

Mirror nodes work by validating the signature files associated with the record, balance, and event files from the consensus nodes that were uploaded to a cloud storage solution from the network.

As transactions reach consensus on the Hedera network, either mainnet or testnet, Hedera consensus nodes add the transaction and its associated records to a record file. A record file contains a series of ordered transactions and their associated records. After a given amount of time, a record file is closed and a new one is created. This process repeats as the network continues to receive transactions.

Once a record file is closed, the consensus nodes generate a signature file. The signature file contains a signature for the corresponding record file's hash. Along with the signature file of the consensus node, the record file also contains the hash of the previous record file. The former record file can now be

verified by matching the hash of the previous record file.

Hedera consensus nodes push new record files and signature files to the cloud storage provider – currently AWS S3 and Google File Storage are supported. Mirror nodes download these files, verify their signatures based on their hashes, and only then make them available to be processed.

Smart Contract Synthetic Logs

Starting with v0.79 of Hedera Mirror Node release, synthetic event logs for Hedera Token Service (HTS) token transactions have been introduced to mimic the behavior of smart contract tokens. Synthetic events are generated for transactions such as:

- CryptoTransfer
- CryptoApproveAllowance
- CryptoDeleteAllowance
- TokenMint
- TokenWipe
- TokenBurn

This feature enables developers to effectively monitor HTS token activities as if they were smart contract tokens. An example code implementation demonstrating using ethers.js to listen to synthetic events can be found [here](#);

REST API from Hedera

Hedera provides REST APIs to easily query a mirror node that is hosted by Hedera, removing the complexity of having to run your own. Check out the mirror node REST API docs [below](#).

```
{% content-ref url="../../../sdks-and-apis/rest-api.md" %}  
rest-api.md  
{% endcontent-ref %}
```

Run a Mirror Node

Anyone can run a Hedera Mirror Node by downloading and configuring the software on their computer. By running a mirror node, you are able to connect to the appropriate cloud storage and store account balance files, record files, and event files as described above. Please check out the [below links](#) on how to get started.

```
{% content-ref url="run-your-own-beta-mirror-node/" %}  
run-your-own-beta-mirror-node  
{% endcontent-ref %}  
  
{% content-ref url="one-click-mirror-node-deployment.md" %}  
one-click-mirror-node-deployment.md  
{% endcontent-ref %}
```

FAQ

<details>

<summary>How is data stored in a Hedera Mirror Node? Is it a specific type of database, or does it use a unique data structure?</summary>

Hedera Mirror Nodes use PostgreSQL databases to store the transaction and event data organized in a structure that mirrors the Hedera Network. Once the mirror node receives record files from Hedera Consensus nodes, the data is validated and loaded into the database.

</details>

<details>

<summary>How do I run my own Hedera Mirror Node? What are the hardware and software requirements?</summary>

Setting up a Hedera Mirror Node involves both hardware and software components. The requirements can be found here.

To run your mirror node, follow the steps in the "Run Your Own Mirror Node" guide.

</details>

<details>

<summary>Are there costs associated with running a mirror node?</summary>

No, Hedera does not charge for running a mirror node. However, there are costs associated with purchasing the hardware, internet connection, and potential cloud service fees. The hardware and software requirements can be found here.

</details>

<details>

<summary>How do I configure a mirror node and query data?</summary>

You can configure your own Hedera Mirror Node by following the step-by-step instructions provided in the "How to Configure a Mirror Node and Query Data" guide. The guide provides instructions on prerequisites, node setup, configuration, and querying the node. Additionally, you can find more details about retention and transaction and entity filtering in the guide.

</details>

<details>

<summary>How can I provide feedback or create an issue to log errors?</summary>

To provide feedback or log errors, please refer to the Contributing Guide and submit an issue in the Hedera Docs GitHub repository.

</details>

README.md:

Run Your Own Mirror Node

Overview

A Hedera Mirror Node is a node that receives pre-constructed files from the Hedera Mainnet. These pre-constructed files include transaction records and account balance files. Transaction records include information about a transaction, like the transaction ID, transaction hash, account, etc. The account balance files give you a snapshot of the balances for all accounts at a given timestamp.

In this tutorial, you will run your own Hedera Mirror Node. You will need to create either a Google Cloud Platform (GCP) or an Amazon Web Services (AWS) account if you do not have one already. The Hedera Mirror Node object storage bucket, where you will pull the account balance and transaction data from, is stored in GCP or AWS bucket and is configured for requester pays. This means the

(the latest version) are installed on your machine.

1. Obtain Google Cloud Platform Requester Pay Information

In this step, you will generate your Google Cloud Platform HMAC access keys. These keys are needed to authenticate requests between your machine and Google Cloud Storage. They are similar to a username and password. Follow these steps to retrieve your access key, secret, and project ID:

Create a new project and link your billing account.
From the left navigation bar, select Cloud Storage > Settings.
Click the Interoperability tab and scroll down to the User account HMAC section.

If you don't already have a default project set, set it now.

Click create keys to generate access keys for your account.

<figure></figure><figcaption></figcaption></figure>

You should see the access key and secret columns populate on the access keys table.

You will use these keys to configure the application.yml file in a later step.

2. Clone Hedera Mirror Node Repository

Open your terminal and run the following commands to clone the hedera-mirror-node repository then cd into the hedera-mirror-node folder:

```
bash
git clone https://github.com/hashgraph/hedera-mirror-node
cd hedera-mirror-node
```

3. Configure Mirror Node

The application.yml file is the main configuration file for the Hedera Mirror Node. We'll update that file with your GCP keys and the Hedera Network you want to mirror.

Open the application.yml file in the root directory with a text editor of your choice.

Find the following section and replace the placeholders with your actual GCP access key, secret key, project ID, and the network you want to mirror:

Item	Description
accessKey	Your access key from your GCP account
cloudProvider	GCP
secretKey	Your secret key from your GCP account
gcpProjectId	Your GCP project ID
network	Enter the network you would to run your mirror node for

```
{% code title="application.yml" %}
yaml
hedera:
  mirror:
    importer:
      downloader:
        accessKey: ENTER ACCESS KEY HERE
        cloudProvider: "GCP"
        secretKey: ENTER SECRET KEY HERE
        gcpProjectId: ENTER GCP PROJECT ID HERE
        network: PREVIEWNET/TESTNET/MAINNET #Pick one network
```

```
{% endcode %}
```

Save the changes and close the file.

4. Start Your Hedera Mirror Node

Now let's start the Hedera Mirror Node using Docker. Docker allows you to easily run applications in a self-contained environment called a container.

From the hedera-mirror-node directory, run the following command:

```
bash
docker compose up -d db && docker logs hedera-mirror-node-db-1 --follow
```

5. Access Your Hedera Mirror Node Data

This step shows you how to access the data that your Hedera Mirror Node is collecting. The mirror node stores its data in a PostgreSQL database, and you're using Docker to connect to that database. To access the mirror node data, we'll have to enter the hedera-mirror-node-db-1 container.

Open a new terminal and run the following command to view the list of containers:

```
bash
docker ps
```

Enter the following command to access the Docker container:

```
bash
docker exec -it hedera-mirror-node-db-1 bash
```

Enter the following command to access the database:

```
bash
psql "dbname=mirrornode host=localhost user=mirrornode password=mirrornodepass
port=5432"
```

Enter the following command to view the complete list of database tables:

```
bash
\dt
```


.png)

To exit the psql console, run the quit command:

```
bash
\q
```

Lastly, run the following command to stop and remove the created containers:

```
bash
docker compose down
```

Congratulations! You have successfully run and deployed a Hedera Mirror Node with Google Cloud Storage (GCS) 

run-your-own-mirror-node-s3.md:

Run Your Mirror Node with Amazon Web Services S3 (AWS)

Prerequisites

An Amazon Web Services account.
Basic understanding of Hedera Mirror Nodes.
Docker (>= v20.10.x) installed and opened on your machine. Run `docker -v` in your terminal to check the version you have installed.
Java (openjdk@17: Java version 17), Gradle (the latest version), and PostgreSQL (the latest version) are installed on your machine.

1. Create an IAM user

This step will teach you how to create a new IAM (Identity and Access Management) user and generate new access keys in your AWS account. The access key, secret and project ID will be used to access S3 from the Hedera Mirror Node.

Create an IAM (Identity and Access Management) user with either an administrator or custom policy. If you're unfamiliar with using AWS, go with the administrator policy:

```
{% tabs %}
{% tab title="Administrator Policy" %}
  Refer to AWS documentation to create an IAM user with an administrator policy here
  This sets up an IAM user with Administrator Access Policy
  This user has full access and can delegate permissions to every service and resource in AWS.
  Once that is complete, select Users from the left IAM navigation bar
  Select the Administrator from the User name column
  Select the Security credentials tab
  Select Create access key
  Copy or download your Access key ID and Secret access key
{% endtab %}
```

```
{% tab title="Custom Policy" %}
  Enable access to billing data
  Follow step 2 here
  From the IAM left navigation bar select Policies
  Click on Create policy
  Service
    Enter S3 as your service
  Actions
    Access level
    Select List and Read
  Resources
    Select Specify bucket resource ARN for the GetBucketLocation
    Add ARN
      hedera-mainnet-streams
    Add ARN
      hedera-mainnet-streams/\
  Click Next:Tags
  Click Next: Review
  Enter a name for the policy
  Click Create policy
  From the left navigation bar on the IAM console select User Groups
  Click Create group
  Enter a user group name
  Select the policy that was created in the previous step
```

Click Create Group
Click Users from the IAM console left navigation bar
Click on Add user
Enter username
Select Programmatic access for Access type
Click Next: Permissions
Select the group that was created in the previous step
Click Next: Tags
Click Next: Review
Click Create user
Copy or download your Access key ID and Secret access key
{% endtab %}
{% endtabs %}

2. Clone the Mirror Node Repository

Open your terminal and run the following commands to clone the mirror node repository, then cd into the hedera-mirror-node folder:

```
<pre class="language-bash"><code class="lang-bash"><strong>git clone
https://github.com/hashgraph/hedera-mirror-node.git
</strong>cd hedera-mirror-node
</code></pre>
```

3. Configure Mirror Node

The application.yml file is the main configuration file for the Hedera Mirror Node. In this step, we will update the configuration file with your specific settings, such as your AWS access key, secret, and the Hedera network you want to mirror.

Open the application.yml file in the root directory with a text editor of your choice.

cd into the hedera-mirror-node directory from your terminal or IDE.

Find the following fields and replace the placeholders with your actual AWS access key, secret key, project ID, and the network you want to mirror:

Item	Description
accessKey	AWS access key
cloudProvider	s3
secretKey	AWS secret key
network	Enter a network to run a mirror node for

```
{% code title="application.yml" %}
yaml
hedera:
  mirror:
    importer:
      downloader:
        accessKey: ENTER ACCESS KEY HERE
        cloudProvider: "s3"
        secretKey: ENTER SECRET KEY HERE
        network: PREVIEWNET/TESTNET/MAINNET #Pick one network
```

```
{% endcode %}
```

4. Run Your Mirror Node

Start and run the Hedera Mirror Node using Docker. Docker packages development tools in a self-contained environment called a container that can include libraries, code, runtime, and more.

From the mirror node directory, run the following command:

```
bash
docker compose up -d db && docker logs hedera-mirror-node-db-1 --follow
```

5. Access Your Mirror Node Data

After the mirror node is run successfully, it downloads data from the Hedera Network and stores it in a PostgreSQL database. To access the mirror node data, enter the database container and connect to it using Docker and the `psql` command-line tool.

Open a new terminal and run the following command to view the list of containers:

```
bash
docker ps
```

<figure></figure>

Run the following command to enter the `hedera-mirror-node-db-1` container:

```
bash
docker exec -it hedera-mirror-node-db-1 bash
```

Enter the following command to access and query the database:

```
bash
psql "dbname=mirrornode host=localhost user=mirrornode password=mirrornodepass port=5432"
```

Enter the following command to view the complete list of all the database tables:

```
bash
\d
```

<figure></figure>

To exit the `psql` database console, run the `quit` command:

```
bash
\q
```

Lastly, run the following command to stop Docker and remove the created containers:

```
bash
docker compose down
```

Congratulations! You have successfully run and deployed a Hedera Mirror Node with Amazon Web Services S3 (AWS) 🎉

compiling-smart-contracts.md:

Compiling Smart Contracts

Compiling a smart contract involves using the contract's source code to generate its bytecode and the contract Application Binary Interface (ABI). The Ethereum Virtual Machine (EVM) executes the bytecode to understand and execute the smart contract. Meanwhile, other smart contracts use the ABI to understand how to interact with the deployed contracts on the Hedera network.

<figure><figcaption></figcaption></figure>

Compiling Solidity

The compiler for the Solidity programming language is solc (Solidity Compiler). You can use the compiler directly or embedded in IDEs like Remix IDE or tools like Hardhat and Truffle.

Smart Contract Bytecode

Bytecode is the machine-readable language that the EVM uses to execute smart contracts. The compiler analyzes the code, checks for syntax errors, enforces language-specific rules, and generates the corresponding bytecode.

Example:

This is the example bytecode output, produced in hexadecimal format, when the HelloHedera smart contract source code is compiled.

json

```
608060405234801561001057600080fd5b5060405161055838038061055883398181016040526020
81101561003357600080fd5b81019080805160405193929190846401000000008211156100535760
0080fd5b8382019150602082018581111561006957600080fd5b8251866001820283011164010000
00008211171561008657600080fd5b8083526020830192505050908051906020019080838360005b
838110156100ba57808201518184015260208101905061009f565b50505050905090810190601f16
80156100e75780820380516001836020036101000a031916815260200191505b5060405250505033
6000806101000a81548173fffffffffffffffffffffffffffffffffffffffff021916908373ffffff
fffffffffffffffffffffffffffffffffffffffff1602179055508060019080519060200190610144929190
61014b565b50506101e8565b82805460018160011615610100020316600290049060005260206000
2090601f016020900481019282601f1061018c57805160ff19168380011785556101ba565b828001
600101855582156101ba579182015b828111156101b957825182559160200191906001019061019e
565b5b5090506101c791906101cb565b5090565b5b808211156101e4576000816000905550600101
6101cc565b5090565b610361806101f76000396000f3fe608060405234801561001057600080fd5b
50600436106100365760003560e01c80632e9826021461003b57806332af2edb146100f6575b6000
80fd5b6100f46004803603602081101561005157600080fd5b810190808035906020019064010000
000081111561006e57600080fd5b82018360208201111561008057600080fd5b8035906020019184
60018302840111640100000000831117156100a257600080fd5b91908080601f0160208091040260
20016040519081016040528093929190818152602001838380828437600081840152601f19601f82
0116905080830192505050505050509192919290505050610179565b005b6100fe6101ec565b6040
518080602001828103825283818151815260200191508051906020019080838360005b8381101561
013e578082015181840152602081019050610123565b50505050905090810190601f16801561016b
5780820380516001836020036101000a031916815260200191505b509250505060405180910390f3
5b60008054906101000a900473fffffffffffffffffffffffffffffffffffffffff1673fffffffffff
fffffffffffffffffffffffffffffffffffffffff163373fffffffffffffffffffffffffffffffffffff1614
6101d1576101e9565b80600190805190602001906101e792919061028e565b505b50565b60606001
8054600181600116156101000203166002900480601f016020809104026020016040519081016040
5280929190818152602001828054600181600116156101000203166002900480156102845780601f
1061025957610100808354040283529160200191610284565b820191906000526020600020905b81
548152906001019060200180831161026757829003601f168201915b5050505050905090565b8280
54600181600116156101000203166002900490600052602060002090601f01602090048101928260
1f106102cf57805160ff19168380011785556102fd565b828001600101855582156102fd57918201
5b828111156102fc5782518255916020019190600101906102e1565b5b50905061030a919061030e
565b5090565b5b8082111561032757600081600090555060010161030f565b509056fea264697066
```


73582212201644465f5f73dfd73a518b57770f5adb27f025842235980d7a0f4e15b1acb18e64736f6c63430007000033

Smart Contract Application Binary Interface (ABI)

The ABI is a JSON (JavaScript Object Notation) file that represents the interface definition for the smart contract. It specifies function signatures, input parameters, return types, and other relevant details of the contract's interface. The ABI helps developers understand how to interact with the smart contract in their distributed applications.

Example:

This is the example ABI output produced when the HelloHedera smart contract is compiled.

```
json
{
  "abi": [
    {
      "inputs": [
        {
          "internalType": "string",
          "name": "message",
          "type": "string"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "constructor"
    },
    {
      "inputs": [],
      "name": "getMessage",
      "outputs": [
        {
          "internalType": "string",
          "name": "",
          "type": "string"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "string",
          "name": "message",
          "type": "string"
        }
      ],
      "name": "setMessage",
      "outputs": [],
      "stateMutability": "nonpayable",
      "type": "function"
    }
  ]
}
```

Additional Resources:

Ethereum: Compiling Smart Contracts

System Smart Contracts

System smart contracts are Hedera API functionality logic presented at reserved address locations on the EVM network. These addresses contain reserved function selectors. When a deployed contract calls these selectors, they execute as though a corresponding system contract exists on the network. Both system and user-deployed contracts live at the same address. If a contract is redeployed, it gets a new address while the original address retains the old bytecode.

System Smart Contract Interfaces

Solidity interfaces provide and define a set of functions that other smart contracts can call. The interfaces for all Hedera systems contracts written in Solidity are maintained in the hedera-smart-contracts repository.

{% hint style="warning" %}

Note: The following Solidity examples are not production-ready and are intended solely for instructional purposes to guide developers.

{% endhint %}

The following is a list of available system contracts on Hedera:

Exchange Rate

The exchange rate contract allows you to convert from tinycents to tinybars and from tinybars to tinycents.

Contract Address	Source

0x168	
https://github.com/hashgraph/hedera-smart-contracts/tree/main/contracts/system-contracts/exchange-rate	

Example

<details>

<summary>IExchangeRate.sol</summary>

```
solidity
// SPDX-License-Identifier: Apache-2.0
pragma solidity >=0.4.9 <0.9.0;

interface IExchangeRate {
    // Given a value in tinycents (1e-8 US cents or 1e-10 USD), returns the
    // equivalent value in tinybars (1e-8 HBAR) at the current exchange rate
    // stored in system file 0.0.112.
    //
    // This rate is a weighted median of the the recent" HBAR-USD exchange
    // rate on major exchanges, but should not be treated as a live price
    // oracle! It is important primarily because the network will use it to
    // compute the tinybar fees for the active transaction.
    //
    // So a "self-funding" contract can use this rate to compute how much
    // tinybar its users must send to cover the Hedera fees for the transaction.
    function tinycentsToTinybars(uint256 tinycents) external returns (uint256);
```

```

    // Given a value in tinybars (1e-8 HBAR), returns the equivalent value in
    // tinycents (1e-8 US cents or 1e-10 USD) at the current exchange rate
    // stored in system file 0.0.112.
    //
    // This rate tracks the the HBAR-USD rate on public exchanges, but
    // should not be treated as a live price oracle! This conversion is
    // less likely to be needed than the above conversion from tinycent to
    // tinybars, but we include it for completeness.
    function tinybarsToTinycents(uint256 tinybars) external returns (uint256);
}

```

</details>

Reference: HIP-475.

Hedera Token Service

The Hedera Token Service smart contract precompile provides functions to use the native Hedera Token Service in smart contracts. Tokens created using this method can also be managed using the native Hedera Token Service APIs.

Contract Address	Source
0x167	https://github.com/hashgraph/hedera-smart-contracts/tree/main/contracts/system-contracts/hedera-token-service

Example

<details>

<summary>IHederaTokenService.sol</summary>

```

solidity
// SPDX-License-Identifier: Apache-2.0
pragma solidity >=0.4.9 <0.9.0;
pragma experimental ABIEncoderV2;

interface IHederaTokenService {

    /// Transfers cryptocurrency among two or more accounts by making the
    /// desired adjustments to their
    /// balances. Each transfer list can specify up to 10 adjustments. Each
    /// negative amount is withdrawn
    /// from the corresponding account (a sender), and each positive one is
    /// added to the corresponding
    /// account (a receiver). The amounts list must sum to zero. Each amount is
    /// a number of tinybars
    /// (there are 100,000,000 tinybars in one hbar). If any sender account
    /// fails to have sufficient
    /// hbars, then the entire transaction fails, and none of those transfers
    /// occur, though the
    /// transaction fee is still charged. This transaction must be signed by the
    /// keys for all the sending
    /// accounts, and for any receiving accounts that have receiverSigRequired
    /// == true. The signatures
    /// are in the same order as the accounts, skipping those accounts that
    /// don't need a signature.

```

```

    /// @custom:version 0.3.0 previous version did not include isApproval
    struct AccountAmount {
        // The Account ID, as a solidity address, that sends/receives
        cryptocurrency or tokens
        address accountID;

        // The amount of the lowest denomination of the given token that
        // the account sends(negative) or receives(positive)
        int64 amount;

        // If true then the transfer is expected to be an approved allowance and
        the
        // accountID is expected to be the owner. The default is false
        (omitted).
        bool isApproval;
    }

    /// A sender account, a receiver account, and the serial number of an NFT of
    a Token with
    /// NONFUNGIBLEUNIQUE type. When minting NFTs the sender will be the default
    AccountID instance
    /// (0.0.0 aka 0x0) and when burning NFTs, the receiver will be the default
    AccountID instance.
    /// @custom:version 0.3.0 previous version did not include isApproval
    struct NftTransfer {
        // The solidity address of the sender
        address senderAccountID;

        // The solidity address of the receiver
        address receiverAccountID;

        // The serial number of the NFT
        int64 serialNumber;

        // If true then the transfer is expected to be an approved allowance and
        the
        // accountID is expected to be the owner. The default is false
        (omitted).
        bool isApproval;
    }

    struct TokenTransferList {
        // The ID of the token as a solidity address
        address token;

        // Applicable to tokens of type FUNGIBLECOMMON. Multiple list of
        AccountAmounts, each of which
        // has an account and amount.
        AccountAmount[] transfers;

        // Applicable to tokens of type NONFUNGIBLEUNIQUE. Multiple list of
        NftTransfers, each of
        // which has a sender and receiver account, including the serial number
        of the NFT
        NftTransfer[] nftTransfers;
    }

    struct TransferList {
        // Multiple list of AccountAmounts, each of which has an account and
        amount.
        // Used to transfer hbars between the accounts in the list.
        AccountAmount[] transfers;
    }

```

```

    /// Expiry properties of a Hedera token - second, autoRenewAccount,
autoRenewPeriod
    struct Expiry {
        /// The epoch second at which the token should expire; if an auto-renew
account and period are
        /// specified, this is coerced to the current epoch second plus the
autoRenewPeriod
        int64 second;

        /// ID of an account which will be automatically charged to renew the
token's expiration, at
        /// autoRenewPeriod interval, expressed as a solidity address
        address autoRenewAccount;

        /// The interval at which the auto-renew account will be charged to
extend the token's expiry
        int64 autoRenewPeriod;
    }

    /// A Key can be a public key from either the Ed25519 or ECDSA(secp256k1)
signature schemes, where
    /// in the ECDSA(secp256k1) case we require the 33-byte compressed form of
the public key. We call
    /// these public keys <b>primitive keys</b>.
    /// A Key can also be the ID of a smart contract instance, which is then
authorized to perform any
    /// precompiled contract action that requires this key to sign.
    /// Note that when a Key is a smart contract ID, it <i>doesn't</i> mean the
contract with that ID
    /// will actually create a cryptographic signature. It only means that when
the contract calls a
    /// precompiled contract, the resulting "child transaction" will be
authorized to perform any action
    /// controlled by the Key.
    /// Exactly one of the possible values should be populated in order for the
Key to be valid.
    struct KeyValue {

        /// if set to true, the key of the calling Hedera account will be
inherited as the token key
        bool inheritAccountKey;

        /// smart contract instance that is authorized as if it had signed with a
key
        address contractId;

        /// Ed25519 public key bytes
        bytes ed25519;

        /// Compressed ECDSA(secp256k1) public key bytes
        bytes ECDSAsecp256k1;

        /// A smart contract that, if the recipient of the active message frame,
should be treated
        /// as having signed. (Note this does not mean the <i>code being executed
in the frame</i>
        /// will belong to the given contract, since it could be running another
contract's code via
        /// <tt>delegatecall</tt>. So setting this key is a more permissive
version of setting the
        /// contractID key, which also requires the code in the active message
frame belong to the
        /// the contract with the given id.)
        address delegatableContractId;

```

```

    }

    /// A list of token key types the key should be applied to and the value of
the key
    struct TokenKey {

        /// bit field representing the key type. Keys of all types that have
corresponding bits set to 1
        /// will be created for the token.
        /// 0th bit: adminKey
        /// 1st bit: kycKey
        /// 2nd bit: freezeKey
        /// 3rd bit: wipeKey
        /// 4th bit: supplyKey
        /// 5th bit: feeScheduleKey
        /// 6th bit: pauseKey
        /// 7th bit: ignored
        uint keyType;

        /// the value that will be set to the key type
        KeyValue key;
    }

    /// Basic properties of a Hedera Token - name, symbol, memo,
tokenSupplyType, maxSupply,
    /// treasury, freezeDefault. These properties are related both to Fungible
and NFT token types.
    struct HederaToken {
        /// The publicly visible name of the token. The token name is specified
as a Unicode string.
        /// Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0
byte (NUL).
        string name;

        /// The publicly visible token symbol. The token symbol is specified as a
Unicode string.
        /// Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0
byte (NUL).
        string symbol;

        /// The ID of the account which will act as a treasury for the token as a
solidity address.
        /// This account will receive the specified initial supply or the newly
minted NFTs in
        /// the case for NONFUNGIBLEUNIQUE Type
        address treasury;

        /// The memo associated with the token (UTF-8 encoding max 100 bytes)
        string memo;

        /// IWA compatibility. Specified the token supply type. Defaults to
INFINITE
        bool tokenSupplyType;

        /// IWA Compatibility. Depends on TokenSupplyType. For tokens of type
FUNGIBLECOMMON - the
        /// maximum number of tokens that can be in circulation. For tokens of
type NONFUNGIBLEUNIQUE -
        /// the maximum number of NFTs (serial numbers) that can be minted. This
field can never be changed!
        int64 maxSupply;

        /// The default Freeze status (frozen or unfrozen) of Hedera accounts
relative to this token. If

```

```

        // true, an account must be unfrozen before it can receive the token
        bool freezeDefault;

        // list of keys to set to the token
        TokenKey[] tokenKeys;

        // expiry properties of a Hedera token - second, autoRenewAccount,
autoRenewPeriod
        Expiry expiry;
    }

    /// Additional post creation fungible and non fungible properties of a
Hedera Token.
    struct TokenInfo {
        /// Basic properties of a Hedera Token
        HederaToken token;

        /// The number of tokens (fungible) or serials (non-fungible) of the
token
        int64 totalSupply;

        /// Specifies whether the token is deleted or not
        bool deleted;

        /// Specifies whether the token kyc was defaulted with KycNotApplicable
(true) or Revoked (false)
        bool defaultKycStatus;

        /// Specifies whether the token is currently paused or not
        bool pauseStatus;

        /// The fixed fees collected when transferring the token
        FixedFee[] fixedFees;

        /// The fractional fees collected when transferring the token
        FractionalFee[] fractionalFees;

        /// The royalty fees collected when transferring the token
        RoyaltyFee[] royaltyFees;

        /// The ID of the network ledger
        string ledgerId;
    }

    /// Additional fungible properties of a Hedera Token.
    struct FungibleTokenInfo {
        /// The shared hedera token info
        TokenInfo tokenInfo;

        /// The number of decimal places a token is divisible by
        int32 decimals;
    }

    /// Additional non fungible properties of a Hedera Token.
    struct NonFungibleTokenInfo {
        /// The shared hedera token info
        TokenInfo tokenInfo;

        /// The serial number of the nft
        int64 serialNumber;

        /// The account id specifying the owner of the non fungible token
        address ownerId;
    }

```

```

    /// The epoch second at which the token was created.
    int64 creationTime;

    /// The unique metadata of the NFT
    bytes metadata;

    /// The account id specifying an account that has been granted spending
    permissions on this nft
    address spenderId;
}

    /// A fixed number of units (hbar or token) to assess as a fee during a
    transfer of
    /// units of the token to which this fixed fee is attached. The denomination
    of
    /// the fee depends on the values of tokenId, useHbarsForPayment and
    /// useCurrentTokenForPayment. Exactly one of the values should be set.
    struct FixedFee {

        int64 amount;

        // Specifies ID of token that should be used for fixed fee denomination
        address tokenId;

        // Specifies this fixed fee should be denominated in Hbar
        bool useHbarsForPayment;

        // Specifies this fixed fee should be denominated in the Token currently
        being created
        bool useCurrentTokenForPayment;

        // The ID of the account to receive the custom fee, expressed as a
        solidity address
        address feeCollector;
    }

    /// A fraction of the transferred units of a token to assess as a fee. The
    amount assessed will never
    /// be less than the given minimumAmount, and never greater than the given
    maximumAmount. The
    /// denomination is always units of the token to which this fractional fee
    is attached.
    struct FractionalFee {
        // A rational number's numerator, used to set the amount of a value
        transfer to collect as a custom fee
        int64 numerator;

        // A rational number's denominator, used to set the amount of a value
        transfer to collect as a custom fee
        int64 denominator;

        // The minimum amount to assess
        int64 minimumAmount;

        // The maximum amount to assess (zero implies no maximum)
        int64 maximumAmount;
        bool netOfTransfers;

        // The ID of the account to receive the custom fee, expressed as a
        solidity address
        address feeCollector;
    }

    /// A fee to assess during a transfer that changes ownership of an NFT.

```



```

Defines the fraction of
    /// the fungible value exchanged for an NFT that the ledger should collect
as a royalty. ("Fungible
    /// value" includes both â„¢, and units of fungible HTS tokens.) When the NFT
sender does not receive
    /// any fungible value, the ledger will assess the fallback fee, if present,
to the new NFT owner.
    /// Royalty fees can only be added to tokens of type type NONFUNGIBLEUNIQUE.
    struct RoyaltyFee {
        /// A fraction's numerator of fungible value exchanged for an NFT to
collect as royalty
        int64 numerator;

        /// A fraction's denominator of fungible value exchanged for an NFT to
collect as royalty
        int64 denominator;

        /// If present, the fee to assess to the NFT receiver when no fungible
value
        /// is exchanged with the sender. Consists of:
        /// amount: the amount to charge for the fee
        /// tokenId: Specifies ID of token that should be used for fixed fee
denomination
        /// useHbarsForPayment: Specifies this fee should be denominated in Hbar
        int64 amount;
        address tokenId;
        bool useHbarsForPayment;

        /// The ID of the account to receive the custom fee, expressed as a
solidity address
        address feeCollector;
    }

    /
    Direct HTS Calls
    /

    /// Performs transfers among combinations of tokens and hbars
    /// @param transferList the list of hbar transfers to do
    /// @param tokenTransfers the list of token transfers to do
    /// @custom:version 0.3.0 the signature of the previous version was
cryptoTransfer(TokenTransferList[] memory tokenTransfers)
    function cryptoTransfer(TransferList memory transferList,
TokenTransferList[] memory tokenTransfers)
        external
        returns (int64 responseCode);

    /// Mints an amount of the token to the defined treasury account
    /// @param token The token for which to mint tokens. If token does not
exist, transaction results in
    /// INVALIDTOKENID
    /// @param amount Applicable to tokens of type FUNGIBLECOMMON. The amount to
mint to the Treasury Account.
    /// Amount must be a positive non-zero number represented in
the lowest denomination of the
    /// token. The new supply must be lower than 2^63.
    /// @param metadata Applicable to tokens of type NONFUNGIBLEUNIQUE. A list
of metadata that are being created.
    /// Maximum allowed size of each metadata is 100 bytes
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return newTotalSupply The new supply of tokens. For NFTs it is the
total count of NFTs
    /// @return serialNumbers If the token is an NFT the newly generate serial

```

```

numbers, otherwise empty.
    function mintToken(
        address token,
        int64 amount,
        bytes[] memory metadata
    )
        external
        returns (
            int64 responseCode,
            int64 newTotalSupply,
            int64[] memory serialNumbers
        );

    /// Burns an amount of the token from the defined treasury account
    /// @param token The token for which to burn tokens. If token does not
exist, transaction results in
    ///             INVALIDTOKENID
    /// @param amount Applicable to tokens of type FUNGIBLECOMMON. The amount
to burn from the Treasury Account.
    ///             Amount must be a positive non-zero number, not bigger
than the token balance of the treasury
    ///             account (0; balance], represented in the lowest
denomination.
    /// @param serialNumbers Applicable to tokens of type NONFUNGIBLEUNIQUE. The
list of serial numbers to be burned.
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return newTotalSupply The new supply of tokens. For NFTs it is the
total count of NFTs
    function burnToken(
        address token,
        int64 amount,
        int64[] memory serialNumbers
    ) external returns (int64 responseCode, int64 newTotalSupply);

    /// Associates the provided account with the provided tokens. Must be
signed by the provided
    /// Account's key or called from the accounts contract key
    /// If the provided account is not found, the transaction will resolve to
INVALIDACCOUNTID.
    /// If the provided account has been deleted, the transaction will resolve
to ACCOUNTDELETED.
    /// If any of the provided tokens is not found, the transaction will
resolve to INVALIDTOKENREF.
    /// If any of the provided tokens has been deleted, the transaction will
resolve to TOKENWASDELETED.
    /// If an association between the provided account and any of the tokens
already exists, the
    /// transaction will resolve to TOKENALREADYASSOCIATEDTOACCOUNT.
    /// If the provided account's associations count exceed the constraint of
maximum token associations
    /// per account, the transaction will resolve to
TOKENSPERACCOUNTLIMITEXCEEDED.
    /// On success, associations between the provided account and tokens are
made and the account is
    /// ready to interact with the tokens.
    /// @param account The account to be associated with the provided tokens
    /// @param tokens The tokens to be associated with the provided account. In
the case of NONFUNGIBLEUNIQUE
    ///             Type, once an account is associated, it can hold any
number of NFTs (serial numbers) of that
    ///             token type
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.

```

```

function associateTokens(address account, address[] memory tokens)
    external
    returns (int64 responseCode);

    /// Single-token variant of associateTokens. Will be mapped to a single
entry array call of associateTokens
    /// @param account The account to be associated with the provided token
    /// @param token The token to be associated with the provided account
    function associateToken(address account, address token)
        external
        returns (int64 responseCode);

    /// Dissociates the provided account with the provided tokens. Must be
signed by the provided
    /// Account's key.
    /// If the provided account is not found, the transaction will resolve to
INVALIDACCOUNTID.
    /// If the provided account has been deleted, the transaction will resolve
to ACCOUNTDELETED.
    /// If any of the provided tokens is not found, the transaction will resolve
to INVALIDTOKENREF.
    /// If any of the provided tokens has been deleted, the transaction will
resolve to TOKENWASDELETED.
    /// If an association between the provided account and any of the tokens
does not exist, the
    /// transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.
    /// If a token has not been deleted and has not expired, and the user has a
nonzero balance, the
    /// transaction will resolve to TRANSACTIONREQUIRESZEROTOKENBALANCES.
    /// If a <b>fungible token</b> has expired, the user can disassociate even
if their token balance is
    /// not zero.
    /// If a <b>non fungible token</b> has expired, the user can <b>not</b>
disassociate if their token
    /// balance is not zero. The transaction will resolve to
TRANSACTIONREQUIREDZEROTOKENBALANCES.
    /// On success, associations between the provided account and tokens are
removed.
    /// @param account The account to be dissociated from the provided tokens
    /// @param tokens The tokens to be dissociated from the provided account.
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function dissociateTokens(address account, address[] memory tokens)
        external
        returns (int64 responseCode);

    /// Single-token variant of dissociateTokens. Will be mapped to a single
entry array call of dissociateTokens
    /// @param account The account to be associated with the provided token
    /// @param token The token to be associated with the provided account
    function dissociateToken(address account, address token)
        external
        returns (int64 responseCode);

    /// Creates a Fungible Token with the specified properties
    /// @param token the basic properties of the token being created
    /// @param initialTotalSupply Specifies the initial supply of tokens to be
put in circulation. The
    /// initial supply is sent to the Treasury Account. The supply is in the
lowest denomination possible.
    /// @param decimals the number of decimal places a token is divisible by
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return tokenAddress the created token's address

```

```

function createFungibleToken(
    HederaToken memory token,
    int64 initialTotalSupply,
    int32 decimals
) external payable returns (int64 responseCode, address tokenAddress);

/// Creates a Fungible Token with the specified properties
/// @param token the basic properties of the token being created
/// @param initialTotalSupply Specifies the initial supply of tokens to be
put in circulation. The
/// initial supply is sent to the Treasury Account. The supply is in the
lowest denomination possible.
/// @param decimals the number of decimal places a token is divisible by.
/// @param fixedFees list of fixed fees to apply to the token
/// @param fractionalFees list of fractional fees to apply to the token
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
/// @return tokenAddress the created token's address
function createFungibleTokenWithCustomFees(
    HederaToken memory token,
    int64 initialTotalSupply,
    int32 decimals,
    FixedFee[] memory fixedFees,
    FractionalFee[] memory fractionalFees
) external payable returns (int64 responseCode, address tokenAddress);

/// Creates an Non Fungible Unique Token with the specified properties
/// @param token the basic properties of the token being created
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
/// @return tokenAddress the created token's address
function createNonFungibleToken(HederaToken memory token)
    external
    payable
    returns (int64 responseCode, address tokenAddress);

/// Creates an Non Fungible Unique Token with the specified properties
/// @param token the basic properties of the token being created
/// @param fixedFees list of fixed fees to apply to the token
/// @param royaltyFees list of royalty fees to apply to the token
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
/// @return tokenAddress the created token's address
function createNonFungibleTokenWithCustomFees(
    HederaToken memory token,
    FixedFee[] memory fixedFees,
    RoyaltyFee[] memory royaltyFees
) external payable returns (int64 responseCode, address tokenAddress);

/
  ABIV1 calls
/

/// Initiates a Fungible Token Transfer
/// @param token The ID of the token as a solidity address
/// @param accountId account to do a transfer to/from
/// @param amount The amount from the accountId at the same index
function transferTokens(
    address token,
    address[] memory accountId,
    int64[] memory amount
) external returns (int64 responseCode);

/// Initiates a Non-Fungable Token Transfer

```

```

    /// @param token The ID of the token as a solidity address
    /// @param sender the sender of an nft
    /// @param receiver the receiver of the nft sent by the same index at sender
    /// @param serialNumber the serial number of the nft sent by the same index
at sender
    function transferNFTs(
        address token,
        address[] memory sender,
        address[] memory receiver,
        int64[] memory serialNumber
    ) external returns (int64 responseCode);

    /// Transfers tokens where the calling account/contract is implicitly the
first entry in the token transfer list,
    /// where the amount is the value needed to zero balance the transfers.
Regular signing rules apply for sending
    /// (positive amount) or receiving (negative amount)
    /// @param token The token to transfer to/from
    /// @param sender The sender for the transaction
    /// @param recipient The receiver of the transaction
    /// @param amount Non-negative value to send. a negative value will result
in a failure.
    function transferToken(
        address token,
        address sender,
        address recipient,
        int64 amount
    ) external returns (int64 responseCode);

    /// Transfers tokens where the calling account/contract is implicitly the
first entry in the token transfer list,
    /// where the amount is the value needed to zero balance the transfers.
Regular signing rules apply for sending
    /// (positive amount) or receiving (negative amount)
    /// @param token The token to transfer to/from
    /// @param sender The sender for the transaction
    /// @param recipient The receiver of the transaction
    /// @param serialNumber The serial number of the NFT to transfer.
    function transferNFT(
        address token,
        address sender,
        address recipient,
        int64 serialNumber
    ) external returns (int64 responseCode);

    /// Allows spender to withdraw from your account multiple times, up to the
value amount. If this function is called
    /// again it overwrites the current allowance with value.
    /// Only Applicable to Fungible Tokens
    /// @param token The hedera token address to approve
    /// @param spender the account address authorized to spend
    /// @param amount the amount of tokens authorized to spend.
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function approve(
        address token,
        address spender,
        uint256 amount
    ) external returns (int64 responseCode);

    /// Transfers amount tokens from from to to using the
    /// allowance mechanism. amount is then deducted from the caller's
allowance.
    /// Only applicable to fungible tokens

```

```

    /// @param token The address of the fungible Hedera token to transfer
    /// @param from The account address of the owner of the token, on the behalf
of which to transfer amount tokens
    /// @param to The account address of the receiver of the amount tokens
    /// @param amount The amount of tokens to transfer from from to to
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function transferFrom(address token, address from, address to, uint256
amount) external returns (int64 responseCode);

    /// Returns the amount which spender is still allowed to withdraw from
owner.
    /// Only Applicable to Fungible Tokens
    /// @param token The Hedera token address to check the allowance of
    /// @param owner the owner of the tokens to be spent
    /// @param spender the spender of the tokens
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return allowance The amount which spender is still allowed to withdraw
from owner.
    function allowance(
        address token,
        address owner,
        address spender
    ) external returns (int64 responseCode, uint256 allowance);

    /// Allow or reaffirm the approved address to transfer an NFT the approved
address does not own.
    /// Only Applicable to NFT Tokens
    /// @param token The Hedera NFT token address to approve
    /// @param approved The new approved NFT controller. To revoke approvals
pass in the zero address.
    /// @param serialNumber The NFT serial number to approve
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function approveNFT(
        address token,
        address approved,
        uint256 serialNumber
    ) external returns (int64 responseCode);

    /// Transfers serialNumber of token from from to to using the allowance
mechanism.
    /// Only applicable to NFT tokens
    /// @param token The address of the non-fungible Hedera token to transfer
    /// @param from The account address of the owner of serialNumber of token
    /// @param to The account address of the receiver of serialNumber
    /// @param serialNumber The NFT serial number to transfer
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function transferFromNFT(address token, address from, address to, uint256
serialNumber) external returns (int64 responseCode);

    /// Get the approved address for a single NFT
    /// Only Applicable to NFT Tokens
    /// @param token The Hedera NFT token address to check approval
    /// @param serialNumber The NFT to find the approved address for
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return approved The approved address for this NFT, or the zero address
if there is none
    function getApproved(address token, uint256 serialNumber)
        external
        returns (int64 responseCode, address approved);

```

```

    /// Enable or disable approval for a third party ("operator") to manage
    /// all of msg.sender's assets
    /// @param token The Hedera NFT token address to approve
    /// @param operator Address to add to the set of authorized operators
    /// @param approved True if the operator is approved, false to revoke
approval
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function setApprovalForAll(
        address token,
        address operator,
        bool approved
    ) external returns (int64 responseCode);

    /// Query if an address is an authorized operator for another address
    /// Only Applicable to NFT Tokens
    /// @param token The Hedera NFT token address to approve
    /// @param owner The address that owns the NFTs
    /// @param operator The address that acts on behalf of the owner
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return approved True if operator is an approved operator for owner,
    false otherwise
    function isApprovedForAll(
        address token,
        address owner,
        address operator
    ) external returns (int64 responseCode, bool approved);

    /// Query if token account is frozen
    /// @param token The token address to check
    /// @param account The account address associated with the token
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return frozen True if account is frozen for token
    function isFrozen(address token, address account)
        external
        returns (int64 responseCode, bool frozen);

    /// Query if token account has kyc granted
    /// @param token The token address to check
    /// @param account The account address associated with the token
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return kycGranted True if account has kyc granted for token
    function isKyc(address token, address account)
        external
        returns (int64 responseCode, bool kycGranted);

    /// Operation to delete token
    /// @param token The token address to be deleted
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    function deleteToken(address token) external returns (int64 responseCode);

    /// Query token custom fees
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return fixedFees Set of fixed fees for token
    /// @return fractionalFees Set of fractional fees for token
    /// @return royaltyFees Set of royalty fees for token
    function getTokenCustomFees(address token)

```

```

        external
        returns (int64 responseCode, FixedFee[] memory fixedFees,
FractionalFee[] memory fractionalFees, RoyaltyFee[] memory royaltyFees);

    /// Query token default freeze status
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return defaultFreezeStatus True if token default freeze status is
    frozen.
    function getTokenDefaultFreezeStatus(address token)
        external
        returns (int64 responseCode, bool defaultFreezeStatus);

    /// Query token default kyc status
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return defaultKycStatus True if token default kyc status is
    KycNotApplicable and false if Revoked.
    function getTokenDefaultKycStatus(address token)
        external
        returns (int64 responseCode, bool defaultKycStatus);

    /// Query token expiry info
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return expiry Expiry info for token
    function getTokenExpiryInfo(address token)
        external
        returns (int64 responseCode, Expiry memory expiry);

    /// Query fungible token info
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return fungibleTokenInfo FungibleTokenInfo info for token
    function getFungibleTokenInfo(address token)
        external
        returns (int64 responseCode, FungibleTokenInfo memory
fungibleTokenInfo);

    /// Query token info
    /// @param token The token address to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return tokenInfo TokenInfo info for token
    function getTokenInfo(address token)
        external
        returns (int64 responseCode, TokenInfo memory tokenInfo);

    /// Query token KeyValue
    /// @param token The token address to check
    /// @param keyType The keyType of the desired KeyValue
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return key KeyValue info for key of type keyType
    function getTokenKey(address token, uint keyType)
        external
        returns (int64 responseCode, KeyValue memory key);

    /// Query non fungible token info
    /// @param token The token address to check

```



```

    /// @param serialNumber The NFT serialNumber to check
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return nonFungibleTokenInfo NonFungibleTokenInfo info for token
    serialNumber
    function getNonFungibleTokenInfo(address token, int64 serialNumber)
        external
        returns (int64 responseCode, NonFungibleTokenInfo memory
nonFungibleTokenInfo);

    /// Operation to freeze token account
    /// @param token The token address
    /// @param account The account address to be frozen
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function freezeToken(address token, address account)
        external
        returns (int64 responseCode);

    /// Operation to unfreeze token account
    /// @param token The token address
    /// @param account The account address to be unfrozen
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function unfreezeToken(address token, address account)
        external
        returns (int64 responseCode);

    /// Operation to grant kyc to token account
    /// @param token The token address
    /// @param account The account address to grant kyc
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function grantTokenKyc(address token, address account)
        external
        returns (int64 responseCode);

    /// Operation to revoke kyc to token account
    /// @param token The token address
    /// @param account The account address to revoke kyc
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function revokeTokenKyc(address token, address account)
        external
        returns (int64 responseCode);

    /// Operation to pause token
    /// @param token The token address to be paused
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function pauseToken(address token) external returns (int64 responseCode);

    /// Operation to unpause token
    /// @param token The token address to be unpaused
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function unpauseToken(address token) external returns (int64 responseCode);

    /// Operation to wipe fungible tokens from account
    /// @param token The token address
    /// @param account The account address to revoke kyc
    /// @param amount The number of tokens to wipe
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.

```

```

function wipeTokenAccount(
    address token,
    address account,
    int64 amount
) external returns (int64 responseCode);

/// Operation to wipe non fungible tokens from account
/// @param token The token address
/// @param account The account address to revoke kyc
/// @param serialNumbers The serial numbers of token to wipe
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
function wipeTokenAccountNFT(
    address token,
    address account,
    int64[] memory serialNumbers
) external returns (int64 responseCode);

/// Operation to update token info
/// @param token The token address
/// @param tokenInfo The hedera token info to update token with
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
function updateTokenInfo(address token, HederaToken memory tokenInfo)
    external
    returns (int64 responseCode);

/// Operation to update token expiry info
/// @param token The token address
/// @param expiryInfo The hedera token expiry info
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
function updateTokenExpiryInfo(address token, Expiry memory expiryInfo)
    external
    returns (int64 responseCode);

/// Operation to update token expiry info
/// @param token The token address
/// @param keys The token keys
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
function updateTokenKeys(address token, TokenKey[] memory keys)
    external
    returns (int64 responseCode);

/// Query if valid token found for the given address
/// @param token The token address
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
/// @return isToken True if valid token found for the given address
function isToken(address token)
    external returns
    (int64 responseCode, bool isToken);

/// Query to return the token type for a given address
/// @param token The token address
/// @return responseCode The response code for the status of the request.
SUCCESS is 22.
/// @return tokenType the token type. 0 is FUNGIBLECOMMON, 1 is
NONFUNGIBLEUNIQUE, -1 is UNRECOGNIZED
function getTokenType(address token)
    external returns
    (int64 responseCode, int32 tokenType);

```

```

    /// Initiates a Redirect For Token
    /// @param token The token address
    /// @param encodedFunctionSelector The function selector from the ERC20
interface + the bytes input for the function called
    /// @return responseCode The response code for the status of the request.
SUCCESS is 22.
    /// @return response The result of the call that had been encoded and sent
for execution.
    function redirectForToken(address token, bytes memory
encodedFunctionSelector) external returns (int64 responseCode, bytes memory
response);
}

```

</details>

Reference: HIP-358, HIP-206, HIP-376, HIP-514.

Hedera Account Service

The Hedera Account Service contract provides functions to interact with the Hedera network to manage HBAR allowances.

Contract Address	Source
0x16a	https://github.com/hashgraph/hedera-smart-contracts/tree/main/contracts/system-contracts/hedera-account-service

Example

<details>

<summary>IHederaAccountService.sol </summary>

```

solidity
// SPDX-License-Identifier: Apache-2.0
pragma solidity >=0.4.9 <0.9.0;
pragma experimental ABIEncoderV2;

interface IHederaAccountService {

    /// Returns the amount of hbars that the spender has been authorized to
    spend on behalf of the owner.
    /// @param owner The account that has authorized the spender
    /// @param spender The account that has been authorized by the owner
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    /// @return amount The amount of hbar that the spender has been authorized
    to spend on behalf of the owner.
    function hbarAllowance(address owner, address spender)
    external
    returns (int64 responseCode, int256 amount);

    /// Allows spender to withdraw hbars from the owner account multiple times,
    up to the value amount. If this function is called
    /// again it overwrites the current allowance with the new amount.
    /// @param owner The owner of the hbars
    /// @param spender the account address authorized to spend

```

```

    /// @param amount the amount of hbars authorized to spend.
    /// @return responseCode The response code for the status of the request.
    SUCCESS is 22.
    function hbarApprove(
        address owner,
        address spender,
        int256 amount
    ) external returns (int64 responseCode);
}

```

</details>


Reference: HIP-906, HIP-632.

Pseudo Random Number Generator (PRNG)

The PRNG system contract allows you to generate a pseudo-random number that can be used in smart contracts.

Contract Address	Source

0x169	
https://github.com/hashgraph/hedera-smart-contracts/tree/main/contracts/system-contracts/pseudo-random-number-generator	

Example 

<details>

<summary>IPrngSystemContract.sol</summary>

```

solidity
// SPDX-License-Identifier: Apache-2.0
pragma solidity >=0.4.9 <0.9.0;

interface IPrngSystemContract {
    // Generates a 256-bit pseudorandom seed using the first 256-bits of running
    // hash of n-3 transaction record.
    // Users can generate a pseudorandom number in a specified range using the
    // seed by (integer value of seed % range)
    function getPseudorandomSeed() external returns (bytes32);
}

```

</details>

Reference: HIP-351.

Compiling Smart Contract Example

Hardhat Tutorial

Additional Resources

HTS Precompile Methods

creating-smart-contracts.md:

Creating Smart Contracts

A smart contract is an immutable program consisting of a set of logic (state variables, functions, event handlers, etc.) or rules that can be deployed, stored, and accessed on a distributed ledger technology such as Hedera. The functions contained within a smart contract can update and manage the state of the contract and read data from the deployed contract. They may also create and call other smart contracts functions on the network. Smart contracts are secure, tamper-proof, and transparent, offering a new level of trust and efficiency.

Hedera supports any language that compiles to the Ethereum Mainnet. This includes Solidity and Vyper. These programming languages compile code and produce bytecode that the Ethereum Virtual Machine (EVM) can interpret and understand.

To learn more about the Solidity programming language, check out the documentation maintained by the Solidity team [here](#).

To learn more about Vyper, check out the documentation maintained by the Vyper team [here](#).

In addition, many tools are available to write and compile smart contracts, including the popular Remix IDE and Hardhat. The Remix IDE is a user-friendly platform that allows you to easily write and compile your smart contracts and perform other tasks such as debugging and testing. Using these tools, you can create powerful and secure smart contracts that can be used for various purposes, from simple token transfers to complex financial instruments.

Example

The following is a very simple example of a smart contract written in the Solidity programming language. The smart contract defines the owner and message state variables, along with functions like `setMessage` (which modifies state details by writing) and `getMessage` (which reads state details).

```
solidity
pragma solidity >=0.7.0 <0.8.9;

contract HelloHedera {
    // the contract's owner, set in the constructor
    address owner;

    // the message we're storing
    string message;

    constructor(string memory message) {
        // set the owner of the contract for kill()
        owner = msg.sender;
        message = message;
    }

    function setMessage(string memory message) public {
        // only allow the owner to update the message
        if (msg.sender != owner) return;
        message = message;
    }

    // return a string
    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

Things you should consider when creating a contract

Automatic Token Associations

An auto association slot is one or more slots you approve that allow tokens to be sent to your contract without explicit authorization for each token type. If this property is not set, you must approve each token before it is transferred to the contract for the transfer to be successful via the `TokenAssociateTransaction` in the SDKs. Learn more about auto-token associations [here](#).

This functionality is exclusively accessible when configuring a `ContractCreateTransaction` API through the Hedera SDKs. If you are deploying a contract on Hedera using EVM tools such as Hardhat and the Hedera JSON RPC Relay, please note that this property cannot be configured, as EVM tools lack compatibility with Hedera's unique features.

Admin Key

Contracts have the option to have an admin key. This concept is native to Hedera contracts and allows the contract account properties to be updated. Note that this does not impact the contract bytecode and does not relate to upgradability. If the admin key is not set, you will not be able to update the following Hedera native properties (noted in `ContractUpdateTransactionBody` protobuf) for your contract once it is deployed:

- `autoRenewPeriod`
- `memoField`
- `maxAutomaticTokenAssociations`
- `autoRenewAccountID`
- `stakeID`
- `declineReward`

You cannot set the admin key field if you deploy a contract via tools like Hardhat. This field can be set if desired by deploying a contract using one of the Hedera SDKs.

Max Contract Storage Size

Contracts on Hedera have a storage size limit of 16,384,000 key value pairs (100MB).

Rent

While rent is not enabled for contracts deployed on Hedera today, you will want to be familiar with the concept of rent, as it may potentially impact the costs of maintaining your contract state on the network. Please refer to the [Smart Contract Rent](#) documentation [here](#).

Transaction and Gas Fees

There are Hedera transaction fees and EVM fees associated with deploying a contract. To view the list of base fees, check out the [fees page](#) and the [fee estimator calculator](#) [here](#).

Smart Contract FAQs

<details>

<summary>What is a smart contract?</summary>

A smart contract is a program that is written in a language that can be interpreted by the EVM. Please refer to the glossary for more keywords and definitions.

</details>

<details>

<summary>What programming language does Hedera support for smart contracts?</summary>

Hedera supports Solidity and Vyper.

</details>

<details>

<summary>Can I write and compile my smart contracts using Remix IDE or other Ethereum ecosystem tools? </summary>

You can use Remix IDE or other Ethereum ecosystem tools to write, compile, and deploy your smart contract on Hedera. Check out our EVM-compatible tools [here](#);

</details>

<details>

<summary>Where can I find the smart contracts that are deployed to each Hedera network (previewnet, testnet, mainnet)?</summary>

On your favorite trusted Block Explorer (also called Mirror Node Explorer on Hedera). To view community-hosted explorers check out the network explorer tools page [here](#);

</details>

<details>

<summary>Which ERC token standards are supported on Hedera?</summary>

Hedera supports ERC-20 and ERC-721 token standards and can find the full list of supported standards [here](#).

</details>

hederas-evm-equivalence-goals-and-exceptions.md:

Understanding Hedera for EVM Developers

Hedera vs Ethereum

While Hedera strives for EVM equivalence, it's important to recognize certain unique aspects and fundamental differences in its network architecture and operations, such as the handling of state data structures, hashing algorithms, and the management of accounts and transactions. These distinctions in network behaviors are intentional design choices made to align with EVM standards, thereby achieving EVM compatibility. This approach ensures that while Hedera aligns closely with Ethereum, it also maintains its distinctive features and optimization.

Network and Security Differences

Function	Hedera	Ethereum
Network State Data Structure	Virtual Merkle Tree	Merkle Patricia Trie
Hashing Algorithm	SHA-384	Keccak-256
Security	High security with aBFT	Secure with decentralized PoS network

{% hint style="info" %}

\Note: Hedera's EVM supports Keccak-256. Transactions received through EthereumTransaction (via the JSON-RPC relay) are hashed using Keccak-256. Only transactions using ED25519 keys through the Hedera API (HAPI) are hashed using SHA-384.

{% endhint %}

Account and Authorization Differences

Function	Hedera	Ethereum
Authorization Signatures	Used for transaction authorization outside of smart contracts	Typically used within smart contracts
Special System Accounts	Available with unique properties	Not available
Non-ECDSA Accounts	Non-ECDSA accounts (such as ED or multi-key) are supported by Hedera and	ECDSA accounts are fully compatible
ECDSA accounts	ECDSA accounts are supported by Ethereum and non-ECDSA accounts are not supported	compatible
Account Deletion	Possible	Not possible

Contract and Gas Differences

Function	Hedera	Ethereum
Data Return on Static Calls	Data retrieval must be done through the relay	Data returned directly
Gas Fees	Charges at least 80% of gas fees regardless of transaction outcome	Gas fees depend on transaction outcome but typically 100% of the gas fees are charged and the unused portion is credited back
Contract Lifecycle	Contract entities can expire, rent fees may apply	No expiration or rent fees

Transactions and Queries Differences

Function	Hedera	Ethereum
Transaction Size Limit	6kb	No limit
Transaction Throttling	Transactions may be throttled by gas limits	Transactions pending until future submission
Query Costs	Not free, can use mirror node for free queries	Free read-only calls
Mempools	No mempools available	Mempools available
Cost	Low, predictable fees (fraction of a cent)	Variable, often high gas fees

RPC Endpoint and Communication Differences

Function	Hedera	Ethereum
RPC Block Requests (e.g., <code>ethgetBlockByHash</code> & <code>ethgetBlockByNumber</code>)	Return zero 32bytes hexadecimal value for the <code>stateRoot</code>	Returns the <code>stateRoot</code> hexadecimal value of the final state trie of the block
Communication	Requires communication with both consensus and mirror nodes	Direct communication with nodes

{% hint style="info" %}

Note: Hedera Consensus and mirror nodes do not provide Ethereum RPC API endpoints.

{% endhint %}

Token and Fee Differences

Function	Hedera	Ethereum
Native Tokens	Supports native tokens in addition to ERC-20 and ERC-721 token standards	All ERC token standards but primarily ERC-20 and ERC-721 tokens.
Fee Structure	Complex with two different gas prices	Single gas price
Token Association	Concept of token association	No concept of token association
Keys for Token Functionality	Keys control access to token functionality (<code>KYC</code> , <code>FREEZE</code> , <code>WIPE</code> , supply, fee, and <code>PAUSE</code>)	No equivalent native functionality

{% hint style="info" %}

Note: Token Association only applies to native HTS tokens and does not affect ERC-20/721 tokens.

{% endhint %}

Other Differences

Function	Hedera	Ethereum
Precheck Failures	Multiple precheck failure reasons	Typically single failure reason
HBAR Decimal Precision	8 or 18 (varies across Hedera APIs)	Consistent 18 point decimal precision

README (1).md:

Smart Contracts

README.md:

Smart Contracts

security.md:

Smart Contract Security

The Hedera Smart Contract Service (HSCS) integrates the features of Hedera's third-generation native entity functionality—high throughput, fast finality, predictable and affordable fees, and fair transaction ordering—with a highly optimized and performant second-generation Ethereum Virtual Machine (EVM). We aim to offer comprehensive support for smart contracts originally written for other EVM-compatible chains and to enable their seamless deployment on Hedera.

EVM Equivalence

We strive to ensure that developers can conveniently point to a Hedera-supported RPC endpoint and perform smart contract executions and queries using the same code and similar tools to achieve EVM equivalence. All smart contract transactions are executed using the Besu EVM to realize this objective, and the resulting changes are stored in the Hedera-optimized Virtual Merkle Tree state. Users are thus guaranteed deterministic finality (as opposed to probabilistic finality) of smart contract executions within 2-3 seconds while ensuring that state changes are entirely encompassed within smart contract functionality.

{% hint style="info" %}

🔔 A Comprehensive breakdown of Hedera's EVM equivalence goals and exceptions can be found here.

{% endhint %}

Security Model

Old model (v1) boundaries

The old security model (pre 0.35.2) supported account key signatures provided at transaction time for authorization. Some of the key characteristics of this model included:

- Smart contracts could only change their own storage or the storage they were delegate called with.

- System smart contracts could be delegate called to carry out Hedera Token Service (HTS) operations on behalf of another account - Externally Owned Account (EOA) or contract account.

- Smart Contracts could change an EOA's storage with the appropriate signature in the transaction.

- Smart Contracts could change an EOA's balance with the appropriate signature in the transaction or with prior addition to an allowance approval list.

This greatly improved user experience as contracts could combine transactions in an attempt at atomicity. For instance, a contract could associate, transfer and approve transactions on a user's behalf with one signature. While focusing on usability, this approach did not address cases in which bad actors could carry out an unsanctioned transaction on behalf of a user, e.g., <https://hedera.com/blog/analysis-remediation-of-the-precompile-attack-on-the-hedera-network>

To address this, the core Hedera engineers thoroughly analyzed the Smart Contract Service and the HTS system contracts, aiming to secure the state and token assets of users and the network during Smart Contract executions. The results of this effort are the guidelines in Hedera Services release v0.35.2.

New model (v2) boundaries

In the new security model, account key signatures cannot provide authorization for contract actions. Its key characteristics include:

Smart contracts can only change their own storage or the storage they were delegate called with.

System smart contracts may not be delegate called, except from the Token proxy/facade flow, e.g., HIP 719. In such cases, HTS tokens are represented as smart contracts (see HIP 218) for common ERC methods.

Smart contracts can change an EOAs storage only if the contract ID is contained in the EOAs key.

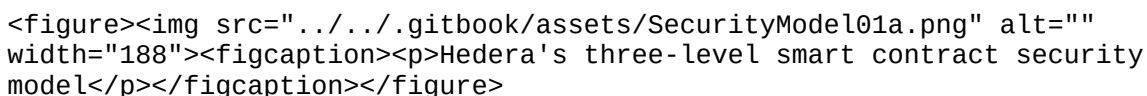
Smart contracts can change an EOAs balance if approved for a token allowance for a specific token held by the EOA.

Boundary comparison table

Boundary Spec	v1 Model	v2 Model
Change	Storage Changes	Smart Contracts could only change their own storage or the storage they were delegate called with
	Smart Contracts could only change their own storage or the storage they were delegate called with	Smart contracts can only change their own storage or the storage they were delegate called with
N	System Contract Call Types	System smart contracts could be delegate called in order to carry out Hedera Token Service operations on behalf of another account (EOA) or contract.
	System smart contracts may not be delegate called, except from the Token facade flow, which presents HTS tokens as smart contracts for common ERC methods.	
Y	Permissioned Account Storage Changes	Smart Contracts could change an EOA's storage with the appropriate signature in the transaction.
	Smart Contracts can change an EOAs storage if the contract ID is contained in the EOAs key.	
Y	Permissioned Account Balance Changes	Smart Contracts could change an accounts (EOA or contract) balance with the appropriate signature in the transaction or with prior addition to an allowance approval list
	Smart contracts can change an EOAs balance if they have been approved a token allowance.	
Y		

In summary, HSCS utilizes a three-level security approach:

1. Level 0 - EVM Security Model: Entities may only modify their own state and balance.
2. Level 1 - ERC Account Value Security Models: Transfer and access to account value will follow tested web3 interface standards, e.g., ERC20, ERC721.
3. Level 2 - Hedera Advanced Security Features: Unique Hedera features may utilize contract-compatible permissions, e.g., ContractID keys.

The diagram illustrates Hedera's three-level smart contract security model. It shows three levels of security: Level 0 (EVM Security Model), Level 1 (ERC Account Value Security Models), and Level 2 (Hedera Advanced Security Features). Each level has specific rules and permissions. Level 0 allows entities to modify their own state and balance. Level 1 allows transfer and access to account value following web3 interface standards. Level 2 allows unique Hedera features like ContractID keys and token allowances.

To achieve state change or value transfer, executions must adhere to the rules of each level. Transactions that don't satisfy the appropriate authorization will fail with response codes such as INVALIDFULLPREFIXSIGNATUREFORPRECOMPILE when a sender is not authorized to carry out an operation. More operational-specific response codes will be returned where applicable e.g. SPENDERDOESNOTHAVEALLOWANCE.

Impact on Developers

As a developer on Hedera, what should I do?

Developers are strongly encouraged to test their applications with new contracts and UX using the new security model to avoid unintended consequences.

The new security model has been applied to contracts created from the mainnet 0.35.2 release and onwards.

Existing contracts deployed before this upgrade will continue to use the previous security model for a limited time to allow for application/UX modifications.

The previous security model will only be maintained for approximately three months. The current target is for the network to remove the previous security model and for all contracts to follow the new model by the mainnet release of July 2023.

See a comprehensive list of the security updates made [here](#).

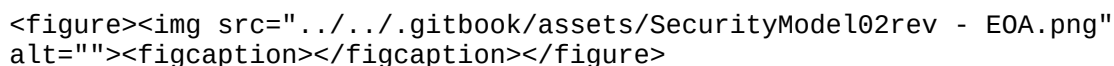
What does the change in the security model mean for smart contract developers?

The security update involves changes to entity permissions during contract executions when modifying the state. In short, system contract calls (smart contract calls to the Hedera Token Service) are no longer executed with all upper caller privileges, even if the authorized user provides a signature.

Understanding the process of contract executions for both externally owned accounts (EOAs) and contracts during regular and delegate calls is crucial. This process involves tracking how accounts, state (storage and value balance), and code may change as you progress through the chain of calls.

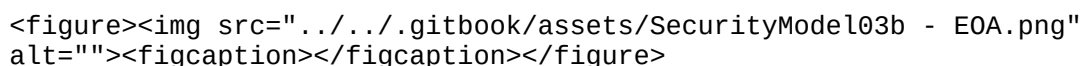
Before (v1 model)

In a regular call scenario, when a call is made to contract B, B's code is executed in the context of its own state. This allows B to modify only its own state. The sender value also differs between the calls to highlight that the EOA made the first call and contract A made the second.



After (v2 model)

On the other hand, in a delegate call scenario, the call to contract B sees B's code executed in the context of A's state. This allows B to modify A's state. The sender and recipient values are preserved from the first call as if the EOA initiated the call.



In summary, a delegate call executes the calling contract's code in the context of the previous account, giving the code access to the previous account's state and blurring the lines of authorized state management.

Applying this to the security model changes, the following table summarizes the authorization check changes.

Scenario	Authorization check	Old Model	New Model
Smart contract A can change its own state using a call	sender = Contract A	Y	Y
Smart contract A can change EOA's state via call	sender = EOA	N	N
Smart contract B can change contract A's state via call	sender = A	N	N
Smart contract A can change EOA's state via delegate call	sender = EOA	Y	Y

Y		Smart contract B can change contract A's state via delegate call	sender = Contract A	Y
Y		System smart contracts can change another accounts (EOA or contract A or contract B) state via call	sender = account	N
N		System smart contract can change another accounts EOA or contract A or contract B) state via delegate call	sender = account	N
N		System contracts can change an accounts (EOA or contact A or contract B) state via call with the appropriate signature		
Y		signature map contains signature of accounts (EOA or contact A or contract B respectively)		
N		System smart contract can change another accounts (EOA or contact A or contract B) state via delegate call with the appropriate signature		
Y		signature map contains signature of accounts (EOA or contact A or contract B)		
N		Contract A or B can call a system contract via a call		Y
Y		Contract A or B can call a system contract via a delegate call		
Y				
N				

At the time of the change, the HTS system contract was the only pathway to expose Hedera API functionality through Smart Contracts. As such, it's fair to consider the differences between pre and post-security model updates when observing HTS system contract state-changing functions.

Existing HTS system contract impacts summary

IHederaTokenService System Smart Contract Function				
v1 Model Authorization Requirements	v2 Model Authorization Requirements	Impacts Code	Solution by Developers	
approve, approveNFT	signature map contains accounts admin key signature	<code>msg.sender</code> must be entity to be modified	Y	<p>Upgrade contracts</p> <p>Upgrade DApps to provide explicit user approval</p> <p>Additional secure pathways: https://hips.hedera.com/hip/hip-376</p> <p>HIP 376</p> <p>IERC.approve()</p>
associateToken	signature map contains account admin key signature	<code>msg.sender</code> must be entity to be modified	Y	<p>Upgrade contracts</p> <p>Upgrade DApps to provide explicit user associate</p> <p>Additional secure pathways: https://hips.hedera.com/hip/hip-719</p> <p>HIP 719</p> <p>IHRC.associate()</p>
burnToken	signature map contains token burn key signature	Contract Id satisfies Token.supplyKey requirements	Y	<p>Contract Id satisfies Token.supplyKey requirements</p> <p>Token admin must set desired contract in Supply key</p> <p>createFungibleToken, createFungibleTokenWithCustomFees, createNonFungibleToken, createNonFungibleTokenWithCustomFees</p> <p>signature map contains affected account admin key signature(s)</p>

in treasury	or	autoRenew assignment case
<p><code>msg.sender</code> must be entity to be modified in treasury</p>		
Y	-	
cryptoTransfer		
signature map contains sender admin key signature		
or	Contract Id satisfies Entity.key requirements	
<p><code>msg.sender</code> must be entity to be modified in treasury</p>	Y	Upgrade DApps to provide explicit user approval.
deleteToken	signature map contains token admin key signature	
	or	Contract Id satisfies Token.adminKey requirements
Contract Id satisfies Token.adminKey requirements	Y	
Token admin must set desired contract in admin key		
dissociateToken, dissociateTokens		
signature map contains admin key signature	<p><code>msg.sender</code> must be entity to be modified</p>	Y
Upgrade contracts	or	Upgrade DApps to provide explicit user dissociate
Additional secure pathways:	https://hips.hedera.com/hip/hip-719	HIP 719
IHRC.associate()		
freezeToken		
signature map contains freeze key signature	or	Contract Id satisfies Token.freezeKey requirements
Contract Id satisfies Token.freezeKey requirements	Y	
Token admin must set desired contract in freeze key		
grantTokenKyc		
signature map contains kyc key signature	or	Contract Id satisfies Token.freezeKey requirements
Contract Id satisfies Token.kycKey requirements	Y	
Token admin must set desired contract in kyc key		
mintToken		
signature map contains appropriate signature	or	Contract Id satisfies Token.supplyKey requirements
Contract Id satisfies Token.supplyKey requirements	Y	
Token admin must set desired contract in Supply key		
pauseToken		
signature map contains pause key signature	or	Contract Id satisfies Token.pauseKey requirements
Contract Id satisfies Token.pauseKey requirements	Y	
Token admin must set desired contract in pause key		
revokeTokenKyc		
signature map contains kyc key signature	or	Contract Id satisfies Token.freezeKey requirements
Contract Id satisfies Token.kycKey requirements	Y	
Token admin must set desired contract in kyc key		
setApprovalForAll		
signature map contains admin key signature		
<p><code>msg.sender</code> must be entity to be modified</p>	Y	Upgrade contracts
Upgrade DApps to provide explicit user associate	Additional secure pathways:	https://hips.hedera.com/hip/hip-376
IERC.setApprovalForAll()		
transferFrom, transferFromNFT		
signature map contains admin key signature	or	Spender must have been pre-approved an allowance

<p><code>msg.sender</code> must be entity to be modified in treasury</p>	<p>or</p>	<p>autoRenew assignment case</p>
Y	Upgrade DApps to provide explicit user approval.	
<p>transferToken, transferTokens, transferNFT, transferNFTs</p>	<p>signature map contains admin key signature</p>	<p>or</p>
	<p>Contract Id satisfies Entity.key requirements</p>	<p>or</p>
	<p>Contract has been approved an allowance to spend by owner</p>	
<p><code>msg.sender</code> must be balance owner.</p>	<p>If not 1. Contract Id satisfies Entity.key requirements</p>	<p>or</p>
	<p>2. Contract has been approved an allowance to spend by owner</p>	
Y	Upgrade DApps to provide explicit user approval.	
<p>updateTokenInfo, updateTokenExpiryInfo, updateTokenKeys</p>	<p>signature map contains token admin key signature</p>	<p>or</p>
	<p>Contract Id satisfies Token.adminKey requirements</p>	<p>or</p>
	<p>Contract Id satisfies Token.adminKey requirements</p>	
Y		
<p>Token admin must set desired contract in admin key</p>	<p>wipeTokenAccount, wipeTokenAccountNFT</p>	<p>signature map contains token wipe key signature</p>
		<p>or</p>
		<p>Contract Id satisfies Token.wipeKey requirements</p>
	<p>Contract Id satisfies Token.wipeKey requirements</p>	<p>Y</p>
	<p>Token admin must set desired contract in Wipe key</p>	
<p>unfreezeToken</p>		
	<p>signature map contains token freeze key signature</p>	<p>or</p>
	<p>Contract Id satisfies Token.freezeKey requirements</p>	<p>or</p>
	<p>Contract Id satisfies Token.freezeKey requirements</p>	
Y		
<p>Token admin must set desired contract in freeze key</p>	<p>unpauseToken</p>	<p>signature map contains token pause key signature</p>
		<p>or</p>
		<p>Contract Id satisfies Token.pauseKey requirements</p>
	<p>Contract Id satisfies Token.pauseKey requirements</p>	<p>Y</p>
	<p>Token admin must set desired contract in pause key</p>	

{% hint style="info" %}

Note: While the changes impact user experience, requiring more explicit steps, they more than proportionately increase user and network security across the board. The team continues to push diligently to provide the community with secure and scalable API solutions to enable them to build creative dApps and carve out their own shared world on the ledger.

{% endhint %}

Security Upgrades

<details>

<summary>0.35.2</summary>

After the security incident on March 9th, the engineers conducted a thorough analysis of the Smart Contract Service and the Hedera Token Service system contracts.

As part of this exercise, we did not find any additional vulnerabilities that could result in an attack that that which we witnessed on March 9th.

The team also looked for any disparities between the expectations of a typical smart contract developer who is used to working with the Ethereum Virtual Machine (EVM) or ERC token APIs and the behaviors of the Hedera Token Service system contract APIs. Such differences in behavior could be used by a malicious smart contract developer in unexpected ways.

In order to eliminate the possibility of these behavioral differences being utilized as attack vectors in the future, the consensus node software will align the behaviors of the Hedera Smart Contract Service token system contracts with those of EVM and typical token APIs such as ERC 20 and ERC 721.

As a result, the following changes are made as of the mainnet 0.35.2 release on March 31st:

An EOA (externally owned account) will have to provide explicit approval/allowance to a contract if they want the contract to transfer value from their account balance.

The behavior of transferFrom system contract will be exactly the same as that of the ERC 20 and ERC 721 spec transferFrom function.

For HTS specific token functionality (e.g. Pause, Freeze, or Grant KYC), a contract will be authorized to perform the associated token management function only if the ContractId is listed as a key on the token (i.e. Pause Key, Freeze Key, KYC Key respectively).

The transferToken and transferNFT APIs will behave as transfer in ERC20/721 if the caller owns the value being transferred, otherwise it will rely on approve spender allowances from the token owner.

The above model will dictate entity (EOA and contracts) permissions during contract executions when modifying state. Contracts will no longer rely on Hedera transaction signature presence, but will instead be in accordance with EVM, ERC and ContractId key models noted.

As part of this release, the network will include logic to grandfather in previous contracts.

Any contracts created from this release onwards will utilize the stricter security model and as such will not have considerations for top-level signatures on transactions to provide permissions.

Existing contracts deployed prior to this upgrade will be automatically grandfathered in and continue to use the old model that was in place prior to this release for a limited time to allow for DApp/UX modification to work with the new security model.

The grandfather logic will be maintained for an approximate period of 3 months from this release. In a future release in July 2023, the network will remove the grandfather logic, and all contracts will follow the new security model.

Developers are encouraged to test their DApps with new contracts and UX using the new security model to avoid unintended consequences. If any DApp developers fail to modify their applications or upgrade their contracts (as applicable) to adhere to the new security model, they may experience issues in their applications.

</details>

smart-contract-addresses.md:

Smart Contract Addresses

After a smart contract is deployed on Hedera, it is associated with a unique smart contract address. There are two types of addresses a smart contract can be referenced by in the system:

☐ Smart Contract EVM Address

☐ Smart Contract ID

EVM Address

The standard smart contract EVM address is the address that is compatible with EVM. The EVM contract address is returned by the system once the contract is deployed. This is the address format that is commonly used in the Ethereum ecosystem. You can use the smart contract EVM address to reference smart

[illegible]

Note: Contracts deployed using the ContractCreate Hedera API transactions will have this form (For example, using ContractCreateTransaction in the SDKs). All other deployment cases will be in the standard EVM address, post HIP-729.

```
{% endhint %}
```

In the Hedera Network, smart contracts can also be identified by a smart contract ID. A smart contract ID is a contract identifier native to the Hedera network. Both the smart contract EVM address and smart contract ID are accepted identifiers for a smart contract when interacting with the contract on Hedera using the Hedera transactions.



The smart contract ID is not a compatible address format accepted or known in the Ethereum ecosystem. For example, if you use MetaMask, you will not specify the contract by its contract ID and instead use its EVM address.

```
<figure><figcaption><p>EVM address &#x26; contract ID example on HashScan</p></figcaption></figure>
```

Similar to Ethereum, Smart Contract entities are also a type of account. A smart contract deployed on Hedera can hold HBAR, fungible, and non-fungible tokens.


Smart Contract Rent


{% hint style="danger" %}

 HEDERA COUNCIL HAS NOT ENABLED RENTS ON SMART CONTRACTS YET. RENTS PAY FOR THE ONGOING USAGE OF RESOURCES USED BY THE SMART CONTRACT. HEDERA INTENDS TO ENABLE THE RENTS IN THE FUTURE, AS DESCRIBED IN THIS SECTION. MORE DETAILS COMING SOON... 

{% endhint %}

Smart contract rent is a recurring payment mechanism designed to maintain resource allocation and is required for contracts to remain active on the network. For contracts, rent is comprised of two primary components:

 Auto-Renewal

 Storage Payments

Contract Auto-Renewal

Auto-renewal is a feature that automatically renews the life of non-deleted smart contracts by a minimum of 90 days. Contract authors are encouraged to establish an auto-renew account specifically for this purpose.

The network will attempt to automatically charge the renewal payment to the expired contract's auto-renew account. The network will attempt to charge the contract if an auto-renew account has zero balance.

If the account lacks sufficient funds for renewal, the contract goes into a one-week grace period. During this time, the contract is inoperable unless funds are added, its expiry is extended (via ContractUpdate), or it receives HBAR. Failing to renew will result in the contract being purged from the state.

Storage Payments

Contract storage payments on Hedera will activate once 100 million key-value pairs are stored cumulatively across the network. The Hedera Coin Economics Committee is expected to set a rate of \$0.02 per key-value pair per year. This applies to all contracts on Hedera, regardless of the contract being created before or after the rent payments go live.

Once storage payments are enabled on Hedera, each contract has 100 free key-value pairs of storage available. Then, once a contract exceeds the first 100 free key-value pairs, it must pay storage fees.

> Storage fees will be part of the rent payment collected when a contract is auto-renewed. Valid renewal windows are between \30 and \92 days (see HIP-372).

If a high enough utilization threshold is reached, congestion pricing applies. In this case, prices charged will be inversely proportional to the remaining system capacity of the network (lower remaining capacity means higher pricing). This applies to all transactions.

Smart Contract Rent - Frequently Asked Questions (FAQ)

<details>

<summary>Why do contracts have to pay rent on Hedera?</summary>

Distributed networks like Hedera have a finite amount of computational

resources. When entities like smart contracts are deployed on a decentralized network, a portion of those resources are consumed. Thus, it is unfeasible to maintain an unlimited number of entities for an infinite amount of time on finite resources. Solving this problem is necessary, and it's a key topic of discussion by Leemon and others in the layer 1 network space.

Contract rent is an economically and technically viable approach to manage smart contract entities and state storage.

</details>

<details>

<summary>Do all entities on Hedera have to pay rent or just contracts?</summary>

All other network entities (e.g., Tokens, accounts, topics, and files) will also pay rent. However, the timeline for the rent is not yet defined. Sufficient time and notice will be provided to the community before enabling rent for other entities.

</details>

<details>

<summary>What charges are included in contract rent?</summary>

Rent is defined as the recurring payment required for contracts (and, eventually, all other Hedera entities) to remain active on the network. For contracts, rent is comprised of auto-renewal and storage payments:

Auto-renewal payments The auto-renewal fee for a contract is \$0.026 USD per 90 days.

Storage payments will start once a total of 100 million key-value pairs are stored cumulatively across the network. These storage fees will be part of the rent payment collected when a contract is auto-renewed. The storage fee rate is \$0.02 per key-value pair per year.

</details>

<details>

<summary>What are the steps in the renewal process? And what happens if a contract doesn't pay rent?</summary>

Every entity on Hedera has the fields `expirationTime`, `autorenewPeriod`, and `autorenewAccount`.

1. When the `expirationTime` for a contract is reached, the network will first try to charge rent to the contract's `autoRenewAccount`

 If renewal is successful, then the contract remains active on the network

 If renewal fails, then the contract is marked as expired

2. An expired entity is given a grace period before it is removed from the network. During the grace period, the entity (contract) is inactive, and all transactions involving it will fail, except for an update transaction to extend the `expirationTime`

 A contract in the grace period can be immediately "re-activated" by either sending it some HBAR or manually extending its `expirationTime` via a contract update transaction

3. At the end of the grace period, the contract is permanently removed from the ledger if:

 The contract and its `autoRenewAccount` still have a zero HBAR balance at the

end of the grace period, OR

The contract is not manually extended during the grace period

Note that the ID number of a removed entity is not reused going forward. In addition, if an entity was marked as deleted, then it cannot have its expirationTime extended. Neither an update transaction nor an auto-renew will be able to extend it.

See the diagram below and HIP-16 for more details.

</details>

<details>

<summary>How long is the grace period for expired contracts?</summary>

The grace period between entity expiration and deletion is 30 days.

</details>

<details>

<summary>Who pays for the contract's renewal and storage fees?</summary>

Smart contracts on Hedera can pay for rent in two ways: external funds or contract funds.

When the expirationTime for a contract is reached, the network will first try to charge rent to the contract's autoRenewAccount:

If the autoRenewAccount has sufficient HBAR to pay for the autoRenewPeriod, then the contract is successfully renewed

If the autoRenewAccount has some HBAR but not enough to afford the full autoRenewPeriod, then the contract is extended for as long as possible (say, 1 week instead of 90 days). Once that extension (1 week) elapses, if the autoRenewAccount hasn't been re-funded to cover the autoRenewPeriod, then the contract account itself will be charged for rent

If the autoRenewAccount has a zero HBAR balance, then the contract itself is charged

If the autoRenewAccount and the contract both have a zero HBAR balance at the time that renewal fees are due, the contract is marked as expired

</details>

<details>

<summary>What happens if I call a contract that is expired?</summary>

Calling an expired contract will resolve to CONTRACTEXPIREDANDAWAITINGREMOVAL.

</details>

<details>

<summary>When a contract is expired and deleted from the network, what happens to its account and assets?</summary>

If an expired contract that holds native Hedera Token Service (HTS) tokens reaches the deletion stage, then the assets held by that contract are returned to their respective treasury accounts.

If the deleted contract is being used as a specific key for an HTS token, then

that key field will refer to a contract that no longer exists. That specific key can be changed, as long as an admin key was specified during token creation. If the token is immutable (no admin key), the specific key cannot be changed.

Contracts that are the treasury for HTS tokens do not expire at this moment (subject to change in the future).

</details>

<details>

<summary>For how long can I renew my contract?</summary>

The minimum renewal period possible is 2,592,000 seconds (\30 days) and the maximum is 8,000,001 seconds (\92 days).

See details in HIP-372: Entity Auto-Renewals and Expiry Window.

</details>

<details>

<summary>If I change the <code>autoRenewPeriod</code> of my contract from 30 to 90 days, what will the cost of my transaction be?</summary>

The cost of rent scales just about linearly with the length of the renewal period. So a renewal that pays for 90 days will cost \3 times as much as a renewal that pays for 30 days.

</details>

<details>

<summary>Where can I see when a contract will expire?</summary>

Mirror nodes provide the expiration time for contracts. You can obtain this information using the mirror node REST API (show it as expirationtime) and network explorers like HashScan (shows it as Expires at).

</details>

<details>

<summary>Where do the auto-renewal transactions appear? Can these be seen on network explorers like HashScan?</summary>

According to HIP-16: Entity Auto-Renewal, records of auto-renew charges will appear as actions in the record stream, and will be available via mirror nodes. In addition, the fee breakdown is provided in network explorers like HashScan for the contract update transaction. No receipts or records for auto-renewal actions will be available via HAPI queries.

HIP-449 provides technical details on how information for expiring contracts is included in the record stream.

</details>

<details>

<summary>Can the <code>autoRenewAccount</code> for a contract be set to another contract ID?</summary>

Yes, that is possible for contracts.

</details>

<details>

<summary>What are the key-value pair thresholds that I should be aware of that impact the size of the storage payment?</summary>

Storage payments for contracts will only start being charged once 100 million key-value pairs are reached cumulatively across the network

After that, each contract has 100 free key-value pairs of storage available. Once a contract exceeds the first 100 free key-value pairs, it must pay storage fees

</details>

<details>

<summary>For smart contracts created via `CREATE2`, how can I specify rent-related properties like `autorenewAccount` and `autorenewPeriod`?</summary>

Contracts created via `CREATE2` inside the EVM will inherit the `autorenewAccount` and `autorenewPeriod` of the sender address.

For example, if you call contract `0xab...cd` which has `autorenewAccount 0.0.X` and `autorenewPeriod` of 45 days, and this contract deploys a new contract `0xcd...ef`, then the new contract will also have `autorenewAccount 0.0.X` and `autorenewPeriod` of 45 days.

Also, remember that rent can be covered by the HBAR balance of a contract. Thus, developers can send HBAR to the contract or configure the contract to charge users a specific HBAR amount when executing operations.

</details>

smart-contract-traceability.md:

Smart Contract Traceability

After contracts have been deployed, you may want to further investigate the execution of a smart contract function call. Traces provide a comprehensive view of the sequence of operations and their effects, allowing for analysis, debugging, and auditing of smart contract behavior. The two types of useful traces:

☐ Call Trace

☐ State Trace

Call Trace

Contract call trace information captures the input, output, and gas details of all the nested smart contracts functions executed in a transaction. On Ethereum, these are occasionally called inner transactions but they simply capture snapshots of the message frame consideration the EVM encounters when processing a smart contract execution at each depth for all involved functions.

Input Data
It records the input data or parameters provided when calling a particular function within a smart contract. This input data is

essentially the encoded form of the function signature and its arguments.

Output Data	After executing the function, the trace information includes the output data returned by that function. This can be the result of the function's computation or any data it generates as part of its execution.
Gas Details	Logs information about the gas consumed by each function call. Each operation within a function consumes a certain amount of gas, and this information is tracked to calculate the overall transaction cost.

This information can be queried using the transaction ID or Ethereum transaction hash.

i Detailed information for call trace can be found in the Hedera protobuf and includes:

Call Trace Data	Description
Call Operation Type	Specific type of operation performed during the execution of a smart contract or a transaction in the EVM. Example: "CALL" is an operation type use when a transaction invokes a function within a smart contract. It executes the function and can potentially modify the state of the contract.
Result Data	The result data is the output or return values generated by the execution of a smart contract function or action. When a function call is executed, it may produce data as a result, such as computed values, status indicators, contract revert reason if any and the error if the transaction itself failed without an explicit
Result Data Type	The "result data type" refers to the data type of the value returned by the function or method. For example, if you have a function <code>add(a, b)</code> that adds two numbers and returns the result, the result data type might be an integer if it returns the sum of the numbers.
Call Depth	The level or depth of the current function call within the call stack. It provides information about the nested nature of function calls and helps track the sequence and hierarchy of function invocations during the execution of a smart contract. The caller depth indicates how many functions have been called before the current function in the call stack. It starts at 0 for the initial function invocation and increments by 1 for each subsequent function call. For example, the parent transaction would be represented as call depth 1 and first child would be at call depth 1.1 and child transaction 2 would be at call depth 1.2. Child transaction at depth 1.2 has two parents.
Caller	The caller can be the ID of the account calling the contract or the ID of another smart contract calling the contract. The first action in the tree can only come from an account. The rest of the actions in the call tree come from the contract. When a smart contract function is invoked, either by an external account or by another contract, the caller address is recorded in the trace to identify the source of the function call. The caller address can be useful in understanding the context of the execution and determining the origin of the transaction or message that triggered the function call.
Recipient	The address of the smart contract or account that receives a specific call or transaction. It represents the destination or target of the interaction within the EVM. The contract action can be directed to one of the following: • Account: The account ID of the recipient if the recipient is an account. Only HBARS will be transferred. • Contract: The contract ID if the recipient is a smart

contract	• EVM address : If the contract action was directed to an invalid solidity address, what that address was	
	From	The from Hedera contract calling the next contract.
	To	The contract receiving the call or being created.
	Value/Amount	The amount of hbars transferred within this call.
	Gas Limit	The gas is defined as the upper limit gas this contract call can spend.
	Gas Used	The amount of gas that was used for the contract call.
	Input	Bytes passed as an input data to this contract call

Example:

```
<figure><figcaption><p>Call trace example on HashScan</p></figcaption></figure>
```

State Trace

Smart Contract state changes will now be tracked whenever a smart contract transaction modifies the state of the contract. This will enable developers to have a paper trail of the state changes that occurred for a contract from the time the contract was created. The state changes that will be tracked include each time a value is read or written to the smart contract. The storage slot represents the order in which the smart contract state is read or written.

The value read reflects the storage value prior to the execution of the smart contract transaction. The value written, if present, represents the final updated value of the storage slot after the completion of the smart contract call. Transient states between the start and finish of the contract are not stored in the transaction record.

i Detailed information on state trace can be found in the protobuf and includes:

[illegible]

Consensus Node

Consensus nodes store sidecar records called `ContractStateChanges`. Each time a smart contract state changes, a new record will be produced that commemorates the state changes for the contract that took place.

Mirror Node

The Hedera mirror node supports two rest APIs that return information about the smart contract's state changes. This includes:


```
/api/v1/contracts/{id}/results/{timestamp}
/api/v1/contracts/results/{transactionIdOrHash}
```

Example:

[illegible]

Hedera Mirror Node Explorer

State trace can be viewed on a supported Hedera Network Explorer.

```
<figure><figcaption><p>State trace example on HashScan</p></figcaption></figure>
```

```
# verifying-smart-contracts-beta.md:
```

Verifying Smart Contracts

Smart contract verification is the process of verifying that the smart contract bytecode uploaded to the network matches the expected smart contract source files. Verification is not required for contracts deployed on the Hedera network, but it is best practice and essential to maintaining the contract's security and integrity by identifying vulnerabilities that could be exploited, as smart contracts are immutable once deployed. It also enables transparency and builds trust within the user community by proving that the deployed bytecode matches the contract's original source code.

To initiate verification, you can use a community-hosted Hedera Mirror Node Explorer, like HashScan (Arkhia and Dragon Glass do not currently support this feature), that integrates with Sourcify: A Solidity source code and metadata verification tool. Once you upload your files to the verification tool, Sourcify recompiles the submitted source code and metadata files to check them against the deployed bytecode. If a match is found, the contract's verification status is updated to either a Full (Perfect) Match or a Partial Match..

The verification status is publicly available across all community-hosted Hedera Mirror Node Explorers. To learn what differentiates a Full (Perfect) Match from a Partial Match, check out the Sourcify documentation [here](#).

```
{% hint style="info" %}
```

Note: This is an initial beta release, and both the HashScan user interface and API functionalities are scheduled for enhancements in upcoming updates.

```
{% endhint %}
```

For verification, you will need the following items:

- ☞ Smart Contract Source Code
- ☞ The Metadata File
- ☞ Deployed Smart Contract Address

Smart Contract Source Code

This is the actual code for your smart contract written in Solidity. The source code includes all the contract's functions, variables, and logic. It's crucial for the verification process, where the deployed bytecode is compared to the compiled bytecode of this source code.

Example:

A simple HelloWorld Solidity smart contract:

```
solidity
pragma solidity ^0.8.17;

contract HelloWorld {
    // the contract's owner, set in the constructor
    address owner;

    // the message we're storing, set in the constructor
    string message;

    constructor(string memory message) {
        // set the owner of the contract for 'kill()'
        owner = msg.sender;
        message = message;
    }

    function setMessage(string memory message) public {
        // only allow the owner to update the message
        if (msg.sender != owner) return;
        message = message;
    }

    // return a string
    function getMessage() public view returns (string memory) {
        return message;
    }
}
```

The Metadata File

When you compile a Solidity smart contract, it generates a JSON metadata file. This file contains settings used when the smart contract was originally compiled. These settings can include the compiler version, optimization details, and more. The metadata file is crucial for ensuring that the bytecode generated during verification matches the deployed bytecode.

> Metadata is not part of the EVM spec because it's handled externally by compilers and tools like Sourcify. See Sourcify's Metadata documentation [here](#);

You have options for generating the metadata file. The recommended skill levels for each option are in parentheses. Choose the option that best fits your

experience with smart contracts:

<details>

<summary>Remix IDE (beginner)</summary>

To create a metadata file in Remix, compile your smart contract and the compiled artifacts will be saved in the artifacts/ directory and the <dynamicid>.json metadata file will be under artifacts/build-info and used for verification. Alternatively, you can copy and paste it from the Solidity compiler tab. Please see the image below.

See the Remix IDE docs for more detailed documentation here.

Note: Taking the bytecode and metadata from Remix and then deploying that on Hedera results in a full (perfect) match. Taking the bytecode and metadata from Remix after deploying the contract on Hedera results in a partial match or The deployed and recompiled bytecode don't match error. The requirement for verification with a contract compiled in Remix is just the smart contract's Solidity file.

</details>

<details>

<summary>Hardhat (intermediate)</summary>

To create the .json metadata file with Hardhat, compile the contract using the npx hardhat compile command. The compiled artifacts will be saved in the artifacts/ directory and the <dynamicid>.json metadata file will be under artifacts/build-info and used for verification. See Sourcify Hardhat metadata documentation here.

Note: The requirement for verification with a contract compiled with Hardhat is only the build-info JSON file.

</details>

<details>

<summary>Foundry (intermediate)</summary>

To create the metadata file with Foundry, compile the contract using the forge build command. The compilation outputs to out/CONTRACTNAME folder. The .json file contains the metadata of the contract under "rawMetadata" and "metadata" fields. However, you don't need to extract the metadata manually for verification. See Sourcify Foundry metadata documentation here.

Note: The requirements for verification with a contract compiled with Foundry are both the .json metadata and the Solidity source file.

</details>

<details>

<summary>Solidity compiler (advanced)</summary>

You can pass the `--metadata` flag to the Solidity command line compiler to get the metadata output printed.

```
solc --metadata contracts/HelloWorld.sol
```

Write the metadata into a file with

```
solc --metadata contracts/HelloWorld.sol > metadata.json
```

Note:solc vs. solcjs

☞ solcjs will not generate the metadata using the `--metadata` flag. The option is only supported in solc.

</details>

An example metadata file for the HelloWorld smart contract:

```
json
{
  "compiler": "0.8.17",
  "language": "Solidity",
  "abi": [
    {
      "inputs": [
        {
          "internalType": "string",
          "name": "message",
          "type": "string"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "constructor"
    },
    {
      "inputs": [],
      "name": "getMessage",
      "outputs": [
        {
          "internalType": "string",
          "name": "",
          "type": "string"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [
        {
          "internalType": "string",
          "name": "message",
          "type": "string"
        }
      ],
      "name": "setMessage",
      "outputs": [],
      "stateMutability": "nonpayable",
      "type": "function"
    }
  ]
}
```

```
]
}
```

Deployed Smart Contract Address

Even though Hedera uses the 0.0.XXXXXXX account ID format, it accommodates Ethereum's address format for EVM compatibility. Once your smart contract is deployed on Hedera's network, you'll receive an address like the one below. This serves as your deployed smart contract address.

Example:

An example deployed EVM smart contract address:

```
0x403925982ef5a6461daba0a103bd6be20b9c4216
```

```
{% hint style="info" %}
```

Note: The 0.0.XXXXXXX smart contract address format can not be used in the verification process.

```
{% endhint %}
```

Different Instances of Sourcify: Hedera's Custom Approach

It's important to note that multiple instances of Sourcify do exist, tailored to the specific needs of different networks. Hedera runs an independent instance of Sourcify, distinct from the public-facing Sourcify.dev instances like Etherscan and other Etherscan clones.

Running an independent instance of Sourcify allows Hedera to have more control over the verification process, tailoring it to the custom needs of the Hedera ecosystem. For instance, after a testnet reset, Hedera requires the ability to reset testnet smart contract verifications - something Sourcify.dev cannot accommodate.

> Verified Smart Contracts Testnet Reset: When the Hedera Testnet is reset, the contract must be redeployed and verified. The contract will receive a new contract EVM address and contract ID. The smart contract will need to be verified using the new addresses.

An essential detail to remember is that smart contracts verified on Hedera's Sourcify instance won't automatically appear as verified on Sourcify.dev or vice versa. Users interested in having their smart contract recognized across multiple platforms should consider verifying on both instances.

Verify Your Smart Contract

Learn how to verify your smart contract:

```
{% content-ref url="../../../tutorials/smart-contracts/how-to-verify-a-smart-contract-on-hashscan.md" %}
```

```
how-to-verify-a-smart-contract-on-hashscan.md
```

```
{% endcontent-ref %}
```

Additional Resources

- ☞ [Sourcify Documentation](#)
- ☞ [HashScan Network Explorer](#)
- ☞ [Smart Contract Verifier Page](#)
- ☞ [Full vs Partial Match Documentation](#)
- ☞ [Hardhat Documentation](#)
- ☞ [Solidity Documentation](#)

gas-and-fees.md:

Gas and Fees

Gas

When executing smart contracts, the EVM requires the amount of work to be paid in gas. The “work” includes computation, state transitions, and storage. Gas is the unit of measurement used to charge a fee per opcode that is executed by the EVM. Each opcode code has a defined gas cost. Gas reflects the cost necessary to pay for the computational resources used to process transactions.

Weibars

The EVM returns gas information in Weibars (introduced in HIP-410). One weibar is 10^{-18} HBAR, which translates to 1 tinybar is 10^{10} weibars. As noted in HIP-410, this is to maximize compatibility with third-party tools that expect ether units to be operated on in fractions of 10^{18} , also known as a Wei.

Gas Schedule and Fees

Gas fees paid for EVM transactions on Hedera can be composed of three different kinds of gas costs:

- Intrinsic Gas
- EVM opcode Gas
- Hedera System Contract Gas

Gas Fee Type	Description
Intrinsic Gas	The minimum amount of gas required to execute a transaction. It is a fixed gas cost that is independent of the specific operations or computations performed within the transaction. Intrinsic gas cost: 21,000 gas
EVM Operation Code	The gas required to execute the defined operation code(s) for the smart contract call. <ul style="list-style-type: none">Opcode Fixed Execution Cost: Each opcode has a fixed cost to be paid upon execution, measured in gas. This cost is the same for all executions, though this is subject to change in new hard forks.Opcode Dynamic Execution Cost: Some instructions conduct more work than others, depending on their parameters. Because of this, on top of fixed costs, some instructions have dynamic costs. These dynamic costs are dependent on several factors (which vary from hard fork to hard fork). See the reference to learn about the specific costs per opcode and fork.
Hedera System Contract Transaction	The required gas that is associated with a Hedera-defined transaction like using the Hedera Token Service system contract that allows you to burn (<code>TokenBurnTransaction</code>) or mint (<code>TokenMintTransaction</code>) a token.

system contract that maps to one of the native Hedera services, you do not need to apply this fee.

The Hedera transaction gas calculation is: Cost of the transaction in USD x Gas Conversion gas/USD + 20%

Example System Contracts:

- Hedera Token Service
- Pseudo Random Number Generator (PRNG)
- Exchange Rate

Gas Limit

The gas limit is the maximum amount of gas you are willing to pay for an operation.

The current opcode gas fees are reflective of the 0.22 Hedera Service release.

Operation	Current Hedera (Gas)
Cancun Cost (Gas)	
-----	-----
Code deposit	
200 \ bytes	200 \ bytes
<code>BALANCE</code> (cold account)	2600
2600	
<code>BALANCE</code> (warm account)	100
100	
EXP	10 +
50/byte	10 + 50/byte
<code>EXTCODECOPY</code> (cold account)	2600
+ Mem	2600 + Mem
<code>EXTCODECOPY</code> (warm account)	100
+ Mem	100 + Mem
<code>EXTCODEHASH</code> (cold account)	2600
2600	
<code>EXTCODEHASH</code> (warm account)	100
100	
<code>EXTCODESIZE</code> (cold account)	2600
2600	
<code>EXTCODESIZE</code> (warm account)	100
100	
<code>LOG0, LOG1, LOG2,</code> <code>LOG3, LOG4</code>	
<p>375 + 375topics + data Mem</p>	<p>375 + 375topics + data Mem</p>
<code>SLOAD</code> (cold slot)	2100
2100	
<code>SLOAD</code> (warm slot)	100
100	
<code>SSTORE</code> (new slot)	
22,100	22,100
<code>SSTORE</code> (existing slot, cold access)	
2,900	2,900
<code>SSTORE</code> (existing slot, warm access)	100
100	
<code>SSTORE</code> refund	As
specified by the EVM	As specified by the EVM
<code>CALL</code> et al. (cold recipient)	
2,600	2,600

	<p><code>CALL</code> et al. (warm recipient)</p>	100
	100	
	<p><code>CALL</code> et al. Hbar/Eth Transfer Surcharge</p>	
9,000	9,000	
	<p><code>SELFDESTRUCT</code> (cold beneficiary)</p>	2600
	2600	
	<p><code>SELFDESTRUCT</code> (warm beneficiary)</p>	0
	0	
	TSTORE	100
	100	
	TLOAD	100
	100	
	MCOPY	3 + 3\
words\copied + memory\expansion\cost	3 + 3\words\copied + memory\expansion\cost	

The terms 'warm' and 'cold' in the above table correspond with whether the account or storage slot has been read or written to within the current smart contract transaction, even if within a child call frame.

'CALL et al.' includes with limitation: CALL, CALLCODE, DELEGATECALL, and STATICCALL

Reference: HIP-206, HIP-865

Gas Per Second Throttling

Most EVM-compatible networks place a gas limit per block to manage resource allocation. This is done to place a limit on the amount of time spent in block validation so that the miner nodes can produce new nodes quickly. While Hedera does not have blocks or miners, in the context of how a Nakamoto consensus system would use it, we are constrained by the physics of time as to how many blocks we can process.

For smart contract transactions, gas is a better measure of the complexity of the EVM transaction than counting all transactions the same, so metering the limits on gas provides a more reasonable limit on resource consumption.

To allow for more flexibility in what transactions we accept and to mirror Ethereum Mainnet behavior, the transaction limits will be calculated on a per-gas basis for smart contract calls (ContractCreate, ContractCall, ContractCallLocalQuery) in addition to a per-transaction limit. This dual approach allows for better resource management, providing a nuanced way to regulate smart contract executions.

The Hedera network has implemented a system gas throttle of 15 million gas per second in the Hedera Service release 0.22.

Gas Reservation and Unused Gas Refund

Hedera throttles transactions prior to consensus, and nodes limit the number of transactions they can submit to the network. Then, at consensus time, if the maximum number of transactions is exceeded, the excess transactions are not evaluated and are canceled with a busy state. Throttling by variable gas amounts provides some challenges to this system, where the nodes only submit a share of their transaction limit.

To address this, throttling will be based on a two-tiered gas measuring system: pre-consensus and post-consensus. Pre-consensus throttling will use the gasLimit field specified in the transaction. Post-consensus will use the actual evaluated amount of gas consumed by the transaction, allowing for dynamic adjustments in the system. It is impossible to know the actual evaluated gas pre-consensus because the network state can directly impact the flow of the transaction, which

is why pre-consensus uses the gasLimit field and will be referred to as the gas reservation.

Contract query requests are unique and bypass the consensus stage altogether. These requests are executed solely on the local node that receives them and only influences that specific node's precheck throttle. On the other hand, standard contract transactions go through both the precheck and consensus stages and are subject to both sets of throttle limits. The throttle limits for precheck and consensus may be set to different values.

In order to ensure that the transactions can execute properly, setting a higher gas reservation than will be consumed by execution is common. On Ethereum Mainnet, the entire reservation is charged to the account prior to execution, and the unused portion of the reservation is credited back. However, Ethereum utilizes a memory pool (mempool) and does transaction ordering at block production time, allowing the block limit to be based only on used and not reserved gas.

To help prevent over-reservation, Hedera restricts the amount of unused gas that can be refunded to a maximum of 20% of the original gas reservation. This effectively means that users will be charged for at least 80% of their initial reservation, regardless of actual usage. This rule is designed to incentivize users to make more accurate gas estimates.

For example, if you initially reserve 5 million gas units for creating a smart contract but end up using only 2 million, Hedera will refund you 1 million gas units, i.e., 20% of your initial reservation. This setup aims to balance the network's resource management while incentivizing users to be as accurate as possible in their gas estimations.

Maximum Gas Per Transaction

Because consensus time execution is now limited by actual gas used and not based on a transaction count, raising the gas limit available for each transaction is safe. Prior to gas-based metering, it would be possible for each transaction to consume the maximum gas per transaction without regard to the other transactions, so limits were based on this worst-case scenario. Now that throttling is the aggregate gas used, we can allow each transaction to consume large amounts of gas without concern for an extreme surge.

When a transaction is submitted to a node with a gasLimit that is greater than the per-transaction gas limit, the transaction must be rejected during precheck with a response code of INDIVIDUALTXGASLIMITEXCEEDED. The transaction must not be submitted to consensus.

Gas throttle per contract call and contract create 15 million gas per second.

Reference: HIP-185

json-rpc-relay.md:

JSON-RPC Relay

The Hedera JSON-RPC Relay is an open-source project implementing the Ethereum JSON-RPC standard. It allows developers to interact with Hedera nodes using familiar Ethereum tools, allowing Ethereum developers and users to deploy, query, and execute contracts as they usually would. Check out the interactive OpenRPC Specification and a simple list of endpoints.

HBAR decimal places

The Hedera JSON RPC Relay msg.value uses 18 decimals when it returns HBAR. As a result, the gasPrice value returns 18 decimal places since it is only utilized


from the JSON RPC Relay. Refer to the HBAR page for a list of Hedera APIs and the decimal places they return.




Community Hosted JSON-RPC Relays

Anyone in the community can set up their own JSON RPC relay that applications can use to deploy, query, and execute smart contracts. The list of community-hosted Hedera JSON RPC relays and endpoints for previewnet, testnet, and mainnet can be found in the table below, as well as their associated docs or websites.

JSON-RPC Relay Endpoints

Network	Chain ID	Hashio RPC URL	thirdweb RPC URL
Mainnet	295	https://mainnet.hashio.io/api	https://295.rpc.thirdweb.com
Testnet	296	https://testnet.hashio.io/api	https://296.rpc.thirdweb.com
Previewnet	297	https://previewnet.hashio.io/api	https://297.rpc.thirdweb.com

 Please note: Hashio is For development and testing purposes only. Production use cases are strongly encouraged to use commercial-grade JSON-RPC relays or host their own instance of the Hedera JSON-RPC Relay.

	https://www.hashgraph.com/hashio/
	https://www.arkhia.io/features/#api-services
	https://docs.validationcloud.io/about/hedera/json-rpc-relay-api
	https://thirdweb.com/hedera
	https://www.quicknode.com/docs/hedera
	https://github.com/hashgraph/hedera-json-rpc-relay?tab=readme-ov-file#hedera-json-rpc-relay

tab=readme-ov-file#hedera-json-rpc-relay</td></tr></tbody></table>

{% hint style="info" %}

Note: If you want to add your hosted JSON-RPC relay to this list, please open an issue in the Hedera docs GitHub repository. Please visit the community-hosted websites to review any limitations specific to their instance.

{% endhint %}

{% content-ref url="../../../tutorials/more-tutorials/json-rpc-connections/" %}

json-rpc-connections

{% endcontent-ref %}

FAQ

<details>

<summary>Are there Community hosted relays?</summary>

Hashio

Arkhia

Validation Cloud

QuickNode

</details>

<details>

<summary>How do I connect to the Hedera Network over RPC?</summary>

The configuration guide to connect to the Hedera Network over RPC can be found here.

</details>

<details>

<summary>Where can I find the Hedera JSON-RPC relay endpoints?</summary>

The endpoints for previewnet, testnet, and mainnet can be found on Hashio, accessible through the Hashgraph website. Feel free to join the discussion on Stack Overflow for more questions.

</details>

<details>

<summary>How does Hedera handle decimals in HBAR and gas prices?</summary>

The JSON-RPC Relay msg.value uses 18 decimals when it returns HBAR. The gasPrice value also returns 18 decimal places. Check out the HBAR page for the full list of Hedera APIs and their decimal representation.

</details>

<details>

<summary>How can I contribute or log errors?</summary>

To contribute or log errors, please refer to the Contributing Guide and submit them as issues in the GitHub repository.

</details>

README.md:

Deploying Smart Contracts

After compiling your smart contract, you can deploy it to the Hedera network. The constructor's "init code" includes the contract's entire bytecode. When deploying, the EVM is expected to be supplied with both the smart contract bytecode and the gas required to execute and deploy the contract. Post-deployment, the constructor is removed, leaving only the runtime bytecode for future contract interactions.

- ☐ Hyperledger Besu EVM
- ☐ Cancun Hard Fork
- ☐ Solidity Variables and Opcodes

Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM) is a run-time environment for executing smart contracts written in EVM native programming languages, like Solidity. The source code must be compiled into bytecode for the EVM to execute a given smart contract.

On Hedera, users can interact with the EVM-compatible environment in several ways. They can submit ContractCreate, EthereumTransaction, or make ethsendRawTransaction RPC calls with the contract bytecode directly. These various paths allow developers to deploy and manage smart contracts efficiently.

When the EVM receives the bytecode, it will be further broken down into operation codes (opcodes). The EVM opcodes represent the specific instructions it can perform. Each opcode is one byte and has its own gas cost associated with it. The cost per opcode for the Ethereum Cancun hard fork can be found [here](#).

Smart Contract Opcode Example

solidity

```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0
DUP1 REVERT JUMPDEST POP PUSH1 0x40 MLOAD PUSH2 0x558 CODESIZE SUB DUP1 PUSH2
0x558 DUP4 CODECOPY DUP2 DUP2 ADD PUSH1 0x40 MSTORE PUSH1 0x20 DUP2 LT ISZERO
PUSH2 0x33 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD
PUSH1 0x40 MLOAD SWAP4 SWAP3 SWAP2 SWAP1 DUP5 PUSH5 0x100000000 DUP3 GT ISZERO
PUSH2 0x53 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP4 DUP3 ADD SWAP2 POP PUSH1
0x20 DUP3 ADD DUP6 DUP2 GT ISZERO PUSH2 0x69 JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST DUP3 MLOAD DUP7 PUSH1 0x1 DUP3 MUL DUP4 ADD GT PUSH5 0x100000000 DUP3
GT OR ISZERO PUSH2 0x86 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP1 DUP4 MSTORE
PUSH1 0x20 DUP4 ADD SWAP3 POP POP POP SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD
SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0xBA JUMPI
DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2
0x9F JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND
DUP1 ISZERO PUSH2 0xE7 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20
SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST
POP PUSH1 0x40 MSTORE POP POP CALLER PUSH1 0x0 DUP1 PUSH2 0x100 EXP DUP2
SLOAD DUP2 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND SWAP1
DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND MUL OR SWAP1 SSTORE
POP DUP1 PUSH1 0x1 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH2 0x144 SWAP3
SWAP2 SWAP1 PUSH2 0x14B JUMP JUMPDEST POP POP PUSH2 0x1E8 JUMP JUMPDEST DUP3
DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2
SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 PUSH1 0x1F
```

```

ADD PUSH1 0x20 SWAP1 DIV DUP2 ADD SWAP3 DUP3 PUSH1 0x1F LT PUSH2 0x18C JUMPI
DUP1 MLOAD PUSH1 0xFF NOT AND DUP4 DUP1 ADD OR DUP6 SSTORE PUSH2 0x1BA JUMP
JUMPDEST DUP3 DUP1 ADD PUSH1 0x1 ADD DUP6 SSTORE DUP3 ISZERO PUSH2 0x1BA JUMPI
SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2 GT ISZERO PUSH2 0x1B9 JUMPI DUP3 MLOAD DUP3
SSTORE SWAP2 PUSH1 0x20 ADD SWAP2 SWAP1 PUSH1 0x1 ADD SWAP1 PUSH2 0x19E JUMP
JUMPDEST JUMPDEST POP SWAP1 POP PUSH2 0x1C7 SWAP2 SWAP1 PUSH2 0x1CB JUMP
JUMPDEST POP SWAP1 JUMP JUMPDEST JUMPDEST DUP1 DUP3 GT ISZERO PUSH2 0x1E4 JUMPI
PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP PUSH1 0x1 ADD PUSH2 0x1CC JUMP
JUMPDEST POP SWAP1 JUMP JUMPDEST PUSH2 0x361 DUP1 PUSH2 0x1F7 PUSH1 0x0 CODECOPY
PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO
PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE LT
PUSH2 0x36 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x2E982602 EQ
PUSH2 0x3B JUMPI DUP1 PUSH4 0x32AF2EDB EQ PUSH2 0xF6 JUMPI JUMPDEST PUSH1 0x0
DUP1 REVERT JUMPDEST PUSH2 0xF4 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2
LT ISZERO PUSH2 0x51 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1
DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH5 0x100000000 DUP2 GT ISZERO
PUSH2 0x6E JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP3 ADD DUP4 PUSH1 0x20 DUP3
ADD GT ISZERO PUSH2 0x80 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP1 CALLDATALOAD
SWAP1 PUSH1 0x20 ADD SWAP2 DUP5 PUSH1 0x1 DUP4 MUL DUP5 ADD GT PUSH5 0x100000000
DUP4 GT OR ISZERO PUSH2 0xA2 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST SWAP2 SWAP1
DUP1 DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40
MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP4 SWAP3 SWAP2 SWAP1 DUP2 DUP2
MSTORE PUSH1 0x20 ADD DUP4 DUP4 DUP1 DUP3 DUP5 CALLDATACOPY PUSH1 0x0 DUP2 DUP5
ADD MSTORE PUSH1 0x1F NOT PUSH1 0x1F DUP3 ADD AND SWAP1 POP DUP1 DUP4 ADD SWAP3
POP POP POP POP POP POP POP SWAP2 SWAP3 SWAP2 SWAP3 SWAP1 POP POP POP PUSH2
0x179 JUMP JUMPDEST STOP JUMPDEST PUSH2 0xFE PUSH2 0x1EC JUMP JUMPDEST PUSH1
0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2
MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1
DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x13E JUMPI DUP1
DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0x123
JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1
ISZERO PUSH2 0x16B JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB
PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP
SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1
0x0 DUP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND CALLER PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ PUSH2 0x1D1 JUMPI PUSH2 0x1E9
JUMP JUMPDEST DUP1 PUSH1 0x1 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH2
0x1E7 SWAP3 SWAP2 SWAP1 PUSH2 0x28E JUMP JUMPDEST POP JUMPDEST POP JUMP JUMPDEST
PUSH1 0x60 PUSH1 0x1 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100
MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV
MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3
SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1
0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2
0x284 JUMPI DUP1 PUSH1 0x1F LT PUSH2 0x259 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV
MUL DUP4 MSTORE SWAP2 PUSH1 0x20 ADD SWAP2 PUSH2 0x284 JUMP JUMPDEST DUP3 ADD
SWAP2 SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2
SLOAD DUP2 MSTORE SWAP1 PUSH1 0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2
0x267 JUMPI DUP3 SWAP1 SUB PUSH1 0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP
POP POP SWAP1 POP SWAP1 JUMP JUMPDEST DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1
AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE
PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 PUSH1 0x1F ADD PUSH1 0x20 SWAP1 DIV DUP2
ADD SWAP3 DUP3 PUSH1 0x1F LT PUSH2 0x2CF JUMPI DUP1 MLOAD PUSH1 0xFF NOT AND
DUP4 DUP1 ADD OR DUP6 SSTORE PUSH2 0x2FD JUMP JUMPDEST DUP3 DUP1 ADD PUSH1 0x1
ADD DUP6 SSTORE DUP3 ISZERO PUSH2 0x2FD JUMPI SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2
GT ISZERO PUSH2 0x2FC JUMPI DUP3 MLOAD DUP3 SSTORE SWAP2 PUSH1 0x20 ADD SWAP2
SWAP1 PUSH1 0x1 ADD SWAP1 PUSH2 0x2E1 JUMP JUMPDEST JUMPDEST POP SWAP1 POP PUSH2
0x30A SWAP2 SWAP1 PUSH2 0x30E JUMP JUMPDEST POP SWAP1 JUMP JUMPDEST JUMPDEST
DUP1 DUP3 GT ISZERO PUSH2 0x327 JUMPI PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP
PUSH1 0x1 ADD PUSH2 0x30F JUMP JUMPDEST POP SWAP1 JUMP INVALID LOG2 PUSH5
0x6970667358 0x22 SLT KECCAK256 AND DIFFICULTY CHAINID 0x5F 0x5F PUSH20
0xDFD73A518B57770F5ADB27F025842235980D7A0F 0x4E ISZERO 0xB1 0xAC 0xB1 DUP15
PUSH5 0x736F6C6343 STOP SMOD STOP STOP CALLER

```

Reference: <https://ethervm.io/>

Hyperledger Besu EVM on Hedera

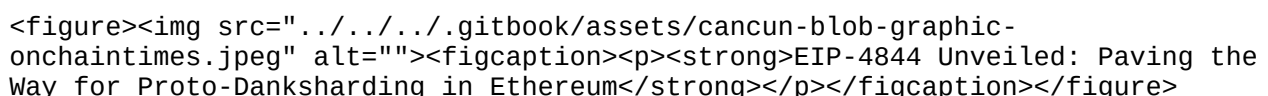
The Hedera network nodes utilize the HyperLedger Besu EVM Client written in Java as an execution layer for Ethereum-type transactions. The codebase is up to date with the current Ethereum Mainnet hard forks. The Besu EVM client library is used without hooks for Ethereum's consensus, networking, and storage features. Instead, Hedera hooks into its own Hashgraph consensus, Gossip communication, and Virtual Merkle Trees components for greater fault tolerance, finality, and scalability.

As of the Hedera Mainnet release 0.50.0, the Besu EVM client is configured to support the Cancun hard fork of the Ethereum Mainnet, with some modifications.

Cancun Hard Fork

The smart contract platform has been upgraded to support the visible EVM changes introduced in the Cancun hard fork. This includes adding new opcodes for transient storage and memory copy, semantic updates for opcodes introduced certain operations introduced in the Shanghai, London, Istanbul, and Berlin hard forks, except those with changes in block production, data serialization, and the double fee market.

As of the Hedera Services 0.22 release, gas and input data costs are charged. The amount of intrinsic gas consumed is a constant charge that occurs before any code executes. The intrinsic gas cost is 21,000. The associated cost of input data is 16 gas for each byte of data that is not zero and 4 gas for each byte of data that is zero. The amount of intrinsic gas consumed is charged in relation to the data supplied when making a contract call to the function parameters of external contracts. The gas schedule and the fees table can be found in the gas section of this documentation page.


<figcaption><p>EIP-4844 Unveiled: Paving the Way for Proto-Danksharding in Ethereum</p></figcaption></figure>

Proto-Danksharding

As an interim solution to full sharding, introduced in the Cancun hard fork, the proto-danksharding offers some of the advantages of sharding with reduced complexity and infrastructure changes that are part of a sharding implementation. This, in turn, opens the gates for adding "blobs" of data to append to blocks to increase data availability further and allow more processing efficiency.

Blobs are big data objects within blocks. These can be utilized to store rollups (Layer 2 solutions) and different kinds of apps requiring big data objects to be stored in an efficient way. This is data off-chain for the validators and requires minimal processing on their part. It reduces the computational load on the network and hence reduces the transaction gas fee.

✗ Blobs supported on Hedera?

Hedera does not provide blobs under EIP-4844. HIP-866 defines how Hedera behaves without blob support. To preserve compatibility and future design space, Hedera will act as if blobs are not being added. This allows existing contracts dependent on blob behavior to function without blobs. Blobs will be prevented from entering the system by prohibiting "Type 3" transactions, which enable blobs. This will keep blobs out of the EVM's concern without affecting other

desirable interactions on Hedera.

Solidity Variables and Opcodes

The table below defines the mapping of Solidity variables and operation codes to Hedera. The full list of supported Opcodes for the Cancun hard fork can be found [here](#);

Solidity	Hedera
<code>address</code>	The address is a mapping of shard.realm.number (0.0.10) into a 20 byte Solidity address. The address can be a Hedera account ID or contract ID in Solidity format.
<code>block.basefee</code>	<code>BASEFEE</code> opcode will return zero. Hedera does not use the Fee Market mechanism this is designed to support.
<code>block.chainId</code>	<code>CHAINID</code> opcode will return 295(hex <code>0x0127</code>) for mainnet, 296(hex <code>0x0128</code>) for testnet, 297(hex <code>0x0129</code>) for previewnet, and 298 (<code>0x12A</code>) for development networks.
<code>block.coinbase</code>	<code>COINBASE</code> operation will return the funding account (Hedera transaction fee collecting account <code>0.0.98</code>).
<code>block.number</code>	The index of the record file (not recommended, use <code>block.timestamp</code>).
<code>block.timestamp</code>	The transaction consensus timestamp.
<code>block.difficulty</code>	Always zero.
<code>block.gaslimit</code>	<code>GASLIMIT</code> operation will return the <code>gasLimit</code> of the transaction. The transaction <code>gasLimit</code> will be the lowest of the gas limit requested in the transaction or a global upper gas limit configured for all smart contracts.
<code>msg.sender</code>	The address of the Hedera contract ID or account ID in Solidity format that called this contract. For the root level or for delegate chains that go to the root, it is the account ID paying for the transaction.
<code>msg.value</code>	The value associated to the transaction associated in tinybar.
<code>tx.origin</code>	The account ID paying for the transaction, regardless of depth.
<code>tx.gasprice</code>	Fixed (varies with the global fee schedule and exchange rate).
<code>selfdestruct</code>	Address will not be reusable due to Hedera's account numbering policies. On <code>SELFDESTRUCT</code> the contracts HBAR and HTS tokens are transferred to the recipients. If the recipient does not exist or does not have an allowance for any of the HTS tokens, this opcode will fail.
<code>&#x3C;address>.code</code>	Precompile contract addresses will report no code, including HTS System contract.
<code>&#x3C;address>.codehash</code>	Precompile contract addresses will report the empty code hash.
<code>PRNGSEED</code>	This opcode returns a random number based on the n-3 record running hash.
<code>delegateCall</code>	

</td><td>Contracts may no longer use <code>delegateCall()</code> to invoke system contracts. Contracts should instead use the <code>call()</code> method.</td></tr><tr><td align="center"><code>blobVersionedHashesAtIndex</code></td><td align="center"><code>BLOBHASH</code></td><td>The <code>BLOBHASH</code> operation will return all zeros at all times.</td></tr><tr><td align="center"><code>blobBaseFee</code></td><td align="center"><code>BLOBBASEFEE</code></td><td><p>The <code>BLOBBASEFEE</code> operation will return</p><p><code>1</code> at all times.</p></td></tr></tbody></table>
--

Reference: HIP-866, HIP-868

Limitation on fallback() / receive() Functions in Hedera Smart Contracts

When developing smart contracts on Hedera, it's important to understand that the fallback() and receive() functions do not get triggered when a contract receives HBAR via a crypto transfer.

In Ethereum, these functions act as "catch-all" mechanisms when a contract receives Ether. In Hedera, however, contract balances may change through native HAPI operations, independent of EVM message calls, making it impossible to maintain balance-related invariants with just the fallback() or receive() methods.

Impacted Variables

msg.sender: The address initiating the contract call.
msg.value: The amount of HBAR sent along with the call.

Key Points

Developers should implement explicit functions to handle HBAR transfers. To disable native operations entirely, consider submitting a Hedera Improvement Proposal (HIP).

Understanding these differences is crucial for anyone developing smart contracts on Hedera, particularly those familiar with Ethereum.

Deploying Your Smart Contract

SDK

You can use a Hedera SDK to deploy your smart contract bytecode to the network. This approach does not require using any EVM tools like Hardhat or an instance of the Hedera JSON-RPC Relay.

```
{% content-ref url="../../../tutorials/smart-contracts/deploy-your-first-smart-contract.md" %}
deploy-your-first-smart-contract.md
{% endcontent-ref %}
```

Hardhat

Hardhat can be used to deploy your smart contract by pointing to a community-hosted JSON-RPC Relay. However, EVM tools do not support features that are native to Hedera smart contracts like:

Admin Key
Contract Memo
Automatic Token Associations
Auto Renew Account ID
Staking Node ID or Account ID
Decline Staking Rewards

If you need to set any of the above properties for your contract, you will have to call the `ContractCreateTransaction` API using one of the Hedera SDKs.

```
{% content-ref url="../../../tutorials/smart-contracts/deploy-a-smart-contract-using-hardhat-hedera-json-rpc-relay.md" %}  
deploy-a-smart-contract-using-hardhat-hedera-json-rpc-relay.md  
{% endcontent-ref %}
```

FAQs

<details>

<summary>Can I use Solidity functions directly with the Hedera EVM?</summary>

Yes, you can use Solidity functions directly with the Hedera EVM. However, refer to the Solidity Variables and Opcodes table to understand any modifications to opcode descriptions that better reflect their behavior on the Hedera network.

</details>

<details>

<summary>What should I do if my contract relies on blob-related opcodes?</summary>

If your contract relies on blob-related opcodes introduced in the Cancun hard fork, you can still deploy it on Hedera. The blob-related opcodes will not fail. They'll return default values as specified by the EVM.

</details>

<details>

<summary>Are there any special considerations for using updated EVM opcodes on Hedera?</summary>

Yes, while the Hedera EVM supports the updated opcodes from the Cancun hard fork, you should know the intrinsic gas costs and input data charges specific to Hedera. Refer to the gas schedule and fees table for more information.

</details>

erc-20-fungible-tokens.md:

ERC-20 (Fungible Tokens)

The ERC-20 standard defines a set of functions and events that a token contract on the Ethereum blockchain should implement. ERC-20 tokens are fungible, meaning each token is identical and can be exchanged one-to-one.

ERC-20 defines several key functions, including:

Supported

<details>

<summary>name</summary>

<mark style="color:purple;">function name() public view returns (string)</mark>

Returns the name of the token.

</details>

<details>

<summary>symbol</summary>

<mark style="color:purple;">function symbol() public view returns (string)</mark>

Returns the symbol of the token.

</details>

<details>

<summary>decimals</summary>

<mark style="color:purple;">function decimals() public view returns (uint8)</mark>

Returns the number of decimals the token uses.

</details>

<details>

<summary>totalSupply</summary>

<mark style="color:purple;">function totalSupply() external view returns (uint256)</mark>

Returns the total supply of the token.

</details>

<details>

<summary>balanceOf</summary>

<mark style="color:purple;">function balanceOf(address account) external view returns (uint256)</mark>

Returns of the balance of the token in the specified account. The <mark style="color:purple;">account</mark> is the Hedera account ID <mark style="color:purple;">0.0.x</mark> in Solidity address format or the evm address of a contract that has been created via the CREATE2 operation.

</details>

<details>

<summary>transfer</summary>

<mark style="color:purple;">function transfer(address recipient, uint256 amount) external returns (bool)</mark>

Transfer tokens from your account to a recipient account. The `<mark style="color:purple;">recipient</mark>` is the Hedera account ID `<mark style="color:purple;">0.0.x</mark>` in Solidity format or the evm address of a contract that has been created via CREATE2 operation.

`</details>`

`<details>`

`<summary>allowance</summary>`

`<mark style="color:purple;">function allowance(address owner, address spender) external view returns (uint256)</mark>`

Returns the remaining number of tokens that spender will be allowed to spend on behalf of owner through transferFrom. This is zero by default. This value changes when approve or transferFrom are called. This works by loading the owner FUNGIBLETOKENALLOWANCES from the accounts ledger and returning the allowance approved for spender. The owner and spender address are the account IDs (0.0.num) in solidity format.

`</details>`

`<details>`

`<summary>approve</summary>`

`<mark style="color:purple;">function approve(address spender, uint256 amount) external returns (bool)</mark>`

Sets amount as the allowance of spender over the caller's tokens.

This works by creating a synthetic CryptoApproveAllowanceTransaction with payer - the account that called the precompile (the message sender property of the message frame in the EVM).

Fires an approval event with the following signature when executed:\n event Approval(address indexed owner, address indexed spender, uint256 value);

`</details>`

`<details>`

`<summary>transferFrom</summary>`

`<mark style="color:purple;">function transferFrom(address sender, address recipient, uint256 amount) external returns (bool)</mark>`

Moves amount tokens from from to to using the allowance mechanism. amount is then deducted from the caller's allowance.

This works by creating a synthetic CryptoTransferTransaction with fungible token transfers with the isapproval property set to true.

`</details>`

Additional References

HIP-376
HIP-218
EIP-20

erc-721-non-fungible-tokens-nfts.md:

ERC-721: Non-Fungible Tokens (NFTs)

The ERC-721 standard introduces a non-fungible token (NFT) in which each issued token is unique and distinct from others. This standard defines functions and events that enable the creation, ownership, and transfer of non-fungible assets.

ERC-721 defines several key functions, including:

Supported

From interface ERC721

<details>

<summary>balanceOf</summary>

<mark style="color:purple;">function balanceOf(address owner) external view
returns (uint256)</mark>

Returns balance of the HTS non fungible token from the account owner. The <mark style="color:purple;">owner</mark> is the Hedera account ID <mark style="color:purple;">0.0.x</mark> in Solidity format or the evm address of a contract that has been created via the CREATE2 operation.

</details>

<details>

<summary>ownerOf</summary>

<mark style="color:purple;">function ownerOf(uint256 tokenId) external view
returns (address)</mark>

Returns the account ID of the specified HTS token owner. The tokenId is the Hedera serial number of the NFT.

</details>

<details>

<summary>approve</summary>

<mark style="color:purple;">function approve(address approved, uint256 tokenId)
external payable</mark>

Gives the spender permission to transfer a token (tokenId) to another account from the owner. The approval is cleared when the token is transferred. The tokenId is the Hedera serial number of the NFT.

This works by creating a synthetic CryptoApproveAllowanceTransaction with payer - the account that called the precompile (the message sender property of the message frame in the EVM).

If the spender address is 0, this creates a CryptoDeleteAllowanceTransaction instead and removes any allowances previously approved on the token.

Fires an approval event with the following signature when executed:

event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);

</details>

<details>

<summary>setApprovalForAll</summary>

<mark style="color:purple;">function setApprovalForAll(address operator, bool approved) external</mark>

Approve or remove an operator as an operator for the caller. Operators can call transferFrom for any token owned by the caller.

This works by creating a synthetic CryptoApproveAllowanceTransaction with payer - the account that called the precompile (the message sender property of the message frame in the EVM).

</details>

<details>

<summary>getApproved</summary>

<mark style="color:purple;">function getApproved(uint256 tokenId) external view returns (address)</mark>

Returns the account approved for the specified tokenId. The tokenId is the Hedera serial number of the NFT.

This works by loading the SPENDER property of the token from the NFTs ledger.

</details>

<details>

<summary>isApprovedForAll</summary>

<mark style="color:purple;">function isApprovedForAll(address owner, address operator) external view returns (bool)</mark>

Returns if the operator is allowed to manage all of the assets of owner.

This works by loading the APPROVEFORALLNFTSALLOWANCES property of the owner account and verifying if the list of approved for all accounts contains the account id of the operator.

</details>

<details>

<summary>transferFrom</summary>

<mark style="color:purple;">function transferFrom(address from, address to, uint256 tokenId) external payable</mark>

Transfers a token (tokenId) from a Hedera account (from) to another Hedera account (to) in Solidity format. The tokenId is the Hedera serial number of the NFT.

This works by creating a synthetic CryptoTransferTransaction with nft token transfers with the isapproval property set to true.

</details>

From interface ERC721Metadata

<details>

<summary>name</summary>

<mark style="color:purple;">function name() external view returns (string name)</mark>

Returns the name of the HTS non-fungible token.

</details>

<details>

<summary>symbol</summary>

<mark style="color:purple;">function symbol() external view returns (string symbol)</mark>

Returns the symbol of the HTS non-fungible token.

</details>

<details>

<summary>tokenURI</summary>

<mark style="color:purple;">function tokenURI(uint256 tokenId) external view returns (string)</mark>

Returns the token metadata of the HTS non-fungible token. This corresponds to the NFT metadata field when minting an NFT using HTS. The tokenId is the Hedera serial number of the NFT.

</details>

From interface ERC721Enumerable

<details>

<summary>totalSupply</summary>

<mark style="color:purple;">function totalSupply() external view returns (uint256)</mark>

Returns the total supply of the HTS non-fungible token.

</details>

Not Supported

The following ERC-721 operations are currently not supported and will return a failure if called.

From interface ERC721

<mark style="color:purple;">safeTransferFrom</mark>

All semantics of interface ERC721TokenReceiver.

Existing Hedera token association rules will take the place of such checks.

From interface ERC721Enumerable

```
<mark style="color:purple;">tokenByIndex</mark>  
<mark style="color:purple;">tokenOfOwnerByIndex</mark>
```

Additional References

HIP-376
HIP-218
EIP-721

hedera-token-service-system-contract.md:

Hedera Token Service System Contract

Hedera enables the native creation of fungible and non-fungible tokens through its SDKs, eliminating the need for smart contracts. This approach leverages Hedera's core features like high TPS, security, and low latency for an optimized user experience. Additionally, the Hedera Token Service provides a cost-effective method for tokenization. Smart contracts on Hedera can also interact with this service via the Hedera Token Service System contract, offering functionalities like token creation, burning, and minting through the EVM.

Some of the key functions defined in the Hedera Token Service System Contract include:

Supported

```
{% @github-files/github-code-block url="https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/system-contracts/hedera-token-service/HederaTokenService.sol" %}
```

Example

```
{% content-ref url="../../../tutorials/smart-contracts/deploy-a-contract-using-the-hedera-token-service.md" %}  
deploy-a-contract-using-the-hedera-token-service.md  
{% endcontent-ref %}
```

Additional References

HIP-206

README.md:

Tokens Managed by Smart Contracts

Smart contracts can be used to create, manage, or serve as the description of tokens. A token is a digital representation of an asset that can include artwork, cryptocurrency, carbon credits, etc.

The ERC-20 and ERC-721 standards define a common interface for token contracts, enabling interoperability between wallets, exchanges, and standardized interaction between different smart contracts and decentralized applications (dApps) in the Ethereum ecosystem. ERC stands for Ethereum Request for Comments, where developers can propose improvements, new features, and protocols for the Ethereum blockchain.

Implementing these interfaces simplifies the process of integrating tokens into applications for developers and ensures consistent user interactions with token contracts. Hedera smart contracts support the following ERCs:

☐ ERC-20 (Fungible Tokens)

☐ ERC-721: Non-Fungible Tokens (NFTs)

Token Associations

Before sending a token to a smart contract, you need to confirm whether you need to associate the token with the smart contract before transferring it. The transfer will fail if you transfer a token to a smart contract that was not associated with it first or does not have an open auto-association slot.

You can associate a smart contract with a token in the following ways:

- Use the `TokenAssociationTransaction` in the supported Hedera SDKs
- Use the `associateToken()` or `associateTokens()` from HIP-206.

{% hint style="info" %}

Note: Token association is for HTS tokens only.

{% endhint %}

Synthetic Events

Smart contract tokens like ERC-20 and ERC-721 emit events, creating contract logs that developers can query or subscribe to. Hedera Token Service (HTS) tokens are not inherently equipped with such event logs. As a solution to this limitation, Hedera Mirror Nodes now generates synthetic event logs for HTS tokens. [Learn more here.](#)

FAQs

<details>

<summary>What should I consider when evaluating managing tokens via a smart contract (EVM) or natively on Hedera for my distributed application?</summary>

Speed: HTS transactions are native and offer faster execution time than a smart contract execution.

Pricing: Native services should be cheaper than the equivalent smart contract scenario.

</details>

<details>

<summary>Do I need to modify my existing contract on another Ethereum chain to use the Hedera token service system contract if my contract adheres to the ERC-720 or ERC-721 standard?</summary>

No, you do not need to modify your existing smart contract deployed to another EVM compatible chain.

</details>

README.md:

Staking

The Hedera public ledger uses a proof-of-stake consensus mechanism, in which each node's influence on consensus is proportional to the amount of cryptocurrency it has staked. A transaction is validated and placed into consensus after it is processed by nodes representing an aggregate stake of over two-thirds of the total amount of HBAR currently staked and dedicated to securing the network. Stake is expressed as an amount in HBAR. It is important to ensure that most of the cryptocurrency is actually being staked, so that the network continues to run. This information can be referenced from the latest Hedera whitepaper.

FAQ

<details>

<summary>What is staking in Hedera?</summary>

Staking is the process of participating in a proof-of-stake system to validate transactions and earn rewards. When staked, coins are locked but can be unlocked for trading. Staking allows participants (stakeholders) to earn rewards on their holdings, typically in tokens or coins.

</details>

<details>

<summary>Is there a lock-up period when accounts are staked to a node?</summary>

No, there is no lock-up period when accounts are staked to a node. The staked account balance is liquid at all times.

</details>

<details>

<summary>How are staking rewards calculated?</summary>

The staking reward rate is determined by the Hedera Governing Council and updated on the mainnet. Learn more about staking rewards [here](#).

</details>

<details>

<summary>How are staking rewards distributed?</summary>

Staking rewards distribution can be triggered by several different mechanisms, such as when an account is staked to a different node, when the total number of HBAR staked to an account changes, or when the staked account is auto-renewed.

</details>

<details>

<summary>Do staking rewards expire?</summary>

Staking rewards do not expire but can only be collected for up to 365 days without a rewards payment being triggered. If more than 365 days pass without a rewards payment, rewards can only be collected for the latest 365 days periods.

</details>

stake-hbar.md:

Stake HBAR

Get Started with Staking



The Hedera Governing Council (via the Coin Committee) votes on the maximum reward rate. The maximum reward can change over time and is not a fixed value. For the latest reward rate, check out the "Nodes" page in HashScan. The actual reward rate will vary depending on how many HBAR are staked for rewards, but the rate will not exceed the cap. In the future, when nodes are down or inactive the staked account will not be eligible to earn rewards.

To view network nodes, their current stake, and reward rate, please visit HashScan.io.

Supported Wallets:

Blade			https://www.bladewallet.io/
HashPack			https://www.hashpack.app/post/staking-with-hashpack
MyHbarWallet			https://myhbarwallet.com/
Uphold			https://uphold.com/en-us/products/staking
Wallawallet			https://wallawallet.com/2022/07/21/how-to-stake-hbar/
Yamgo			https://yamgo.com/

Exchanges:

OKX			https://www.okx.com/support/hc/en-us/articles/9824479181709-Win-up-to-72-APY-with-Hedera-HBAR-Staking
Archax			https://archax.com/custody/hedera-staking

Custodians:

Hex			

Trust				https://hextrust.com/
				hex trust.png

Staking Reward Distribution

Rewards distributions can be triggered by one of the following mechanisms:

- When your staked account has HBAR transferred to it or debited from it
- When you update the staked account to stake to a different node
- When you update the staked account to decline rewards
- When the total number of HBAR staked to an account changes
- When the staked account is auto-renewed (auto-renew for accounts is not enabled at this time)
- When an account staked to this one has its account balance change

You can continue to collect rewards earned for up to 365 days without a rewards payment being triggered

If you go more than 365 days without a rewards payment, you can only collect on the last 365 days

Example: Staker stakes for 1000 days, never collecting a reward, and on the 1001st day collect your rewards

You will only get rewards for the latest 365 periods

You will not earn rewards for the preceding 635 periods (1,000 days - 365 days)

✓ For complete staking program details, check out the Staking Program page.

staking.md:

Staking Program

The staking feature will be rolled out in four phases. The first two phases are described below, and the final two phases will be available at the start of Phase I.

Phase I: Technical Availability \[Complete]

The staking functionality is now available and live on both the Hedera Testnet and Mainnet as of July 21, 2022. In phase I, users will technically be able to stake their account to mainnet nodes but this will not contribute to a node's consensus weight (voting power). This initial technical availability release does not reward participants for staking but enables a level playing field whereby all market participants have the possibility to join the staking program and avoids giving an unfair advantage to the first few who stake.

Phase II: Ecosystem Development \[Complete]

During this phase, supported exchanges and wallets will be able to integrate the staking functionality to provide account holders an easy way to stake their HBAR, but will not distribute rewards. In addition, web applications for delegating stake will likely be built for utilization by the retail ecosystem. During this phase, there will be visibility of stake per node, and staking to a node will affect its consensus weight (voting power) with monthly updates.

Phase III: Staking Rewards Program Launch \[Complete]

The Hedera Governing Council will determine when the Hedera ecosystem has reached a minimum viable set of integrations to enable staking rewards. Once this is determined, the council (through CoinCom) will vote to update the reward rate, and subsequently, the mainnet will be updated with the agreed-upon reward rate. The latest staking reward rate voted on by CoinComm can be found here.\

Once updated, the staking reward account (0.0.800) will be eligible to distribute rewards earned by stakers, once the rewards threshold of 250M total HBAR has been met. Rewards will continue to be distributed even if, after this time, the balance of account 0.0.800 goes below 250M.

Phase IV: Complete Staking Implementation

In this phase, 24-hour updates will be released for visibility into the stake per node, and the node uptime feature will be released. This means that instead of updating node stake visibility on a monthly basis, node stake visibility will be updated on a 24-hour epoch interval. When the uptime feature takes effect, staked accounts will not earn rewards when nodes cannot participate in consensus (unavailable or offline).

Staking Nodes

```
{% hint style="info" %}
```

The Hedera Governing Council voted to change the min stake value from half of the max node stake value to 1/4 of the max node stake value.

```
{% endhint %}
```

All consensus nodes run by the Hedera Governing Council distribute rewards to the accounts staked to them. You can find information about each node in the network by visiting one of the Hedera network explorers or getting the network address book. In the future, network participation will open up to community nodes and eventually to the public as part of Hedera's decentralization efforts.

Nodes have a minimum stake and maximum stake. The node's minimum stake must be met for the accounts staked to that node to be eligible to earn staking rewards. Staked tokens that go over the maximum stake will no longer impact the proportion of rewards returned. The maximum stake threshold for each node will be the total number of HBAR divided by the total number of nodes in the network. The minimum node stake threshold value will be 1/4 of the maximum node stake value. These values will change as more nodes are added to the network or can change by vote of the Hedera Governing Council.

Example:

Minimum Stake: 50,000,000,000 hbars\($\frac{1}{26}$ nodes)\($\frac{1}{4}$)

Maximum Stake: 50,000,000,000 hbars\($\frac{1}{26}$ nodes)

Lockup Period

There is no lock-up period when accounts are staked to a node. Stakers do not need to choose an amount of HBAR to stake from their account. The account's entire balance is staked automatically to the selected node or account. There is no concept of "bonding" or "slashing" of your tokens. The staked account balance is liquid at all times.

Staking Reward Account

The staking reward account distributes rewards to eligible staked accounts. The staking reward account ID is 0.0.800 on mainnet. Anyone in the community can contribute to the rewards pool by transferring HBAR into that account. This account has no keys, and therefore, any HBAR transferred into this account cannot be returned to the owner. If you choose to contribute to the rewards pool, please make sure to double-check your transfer transaction details.

The staking reward account needs to meet a minimum balance before rewards can begin to distribute rewards earned to the eligible staked accounts. The minimum HBAR balance threshold for the reward account is 250 million HBAR voted on by the Hedera Governing Council. If this balance is not met staking rewards will not be distributed. You can view the balance of this account by visiting any of

the Hedera network explorers.

Once the minimum threshold is met, rewards will continue to be distributed to staked accounts as long as there is a balance in the rewards account even if it falls below the initial minimum threshold. The reward rate will initially be set to zero. The Hedera Governing Council will vote and update the reward rate when the Hedera Staking Reward Program goes live. The latest reward rate can be found [here](#);

Staking Rewards

In Phase I, the staking reward rate will initially be zero. The Hedera Governing Council will determine when the Hedera ecosystem has reached a minimum viable set of integrations to enable staking rewards. Once this is determined, the council (through CoinCom) will vote to update the reward rate, and subsequently, the mainnet will be updated with the agreed-upon reward rate.

Any account can elect to stake to a node or another account. The minimum staking period is the minimum amount of time an account needs to be staked to a consensus node before the account is eligible to earn rewards. The minimum staking period is one day (24 hours). The staking period begins at midnight UTC and ends at midnight UTC. The Hedera Governing Council defines the staking period. The earned rewards are not transferred to the staked account immediately after an account has been staked for one full staking period. Please see the Staking Reward Distribution section for what scenarios trigger the payment of a reward.

Accounts staked for less than the defined minimum staking period are not eligible to earn rewards for that period. Nodes and accounts accumulate stake and rewards per whole HBAR. Fractions are rounded down. When a node is deleted, it returns zero rewards.

For a staked account to be eligible to earn rewards, the following must be true:

- The staking reward account needs to have met the initial threshold balance of HBAR

- Once the minimum threshold value has been met, the rewards account will continue to reward staked accounts even if the balance falls below the initial threshold

- The account the node is staked to meets the minimum node stake threshold value


- The account needs to be staked for the minimum staking period

- The reward rate is voted on by the Hedera Governing Council and updated on mainnet

Rewards will continue to be earned when a node is down or inactive in the first phase. The Council (through CoinCom) has voted to implement a maximum cap of 2.5% annual reward rate. The actual reward rate will vary depending on how many HBAR are staked for rewards, but the rate will not exceed the cap. In the future, when nodes are down or inactive the staked account will not be eligible to earn rewards.

This staking system offers an additional unique functionality: indirect staking. If account A stakes to node N, then the stake increases the consensus weight of N, and account A is rewarded for every 24-hour period that it stakes. If account A stakes to account B, and account B stakes to node N, then the stake from both A and B will increase the consensus weight of N, but the rewards for both A and B will be received by B.

An account can optionally decline to earn rewards when staked. The account will still be counted towards meeting the node's minimum stake value.

 If you're interested in checking out the wallets and exchanges supporting staking HBAR, head to the [Stake HBAR](#) page.

create-an-account.md:

Create an Account

Summary

In this section, you will learn how to make a simple Hedera account. Hedera accounts are the entry point by which you can interact with the Hedera APIs. Accounts hold a balance of HBAR used to pay for API calls for the various transaction and query types.

Prerequisites

- Completed the Introduction step.
- Completed the Environment Setup step.

Step 1: Import modules

Import the following modules to your code file.

```
{% tabs %}
{% tab title="Java" %}
java
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.HederaPreCheckStatusException;
import com.hedera.hashgraph.sdk.HederaReceiptStatusException;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.Client;
import com.hedera.hashgraph.sdk.TransactionResponse;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;
import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.AccountBalance;
import io.github.cdimascio.dotenv.Dotenv;
import java.util.concurrent.TimeoutException;

{% endtab %}

{% tab title="JavaScript" %}
javascript
const { Client, PrivateKey, AccountCreateTransaction, AccountBalanceQuery,
TransferTransaction, Hbar } = require("@hashgraph/sdk");
require("dotenv").config();

{% endtab %}

{% tab title="Go" %}
go
import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

{% endtab %}
{% endtabs %}
```

Step 2: Generate keys for the new account

Generate a private and public key to associate with the account you will create.

```
{% tabs %}
{% tab title="Java" %}
java
//Create your Hedera Testnet client
//Client client = Client.forTestnet();
//client.setOperator(myAccountId, myPrivateKey)
//-----<enter code
below>-----

// Generate a new key pair
PrivateKey newAccountPrivateKey = PrivateKey.generateED25519();
PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

{% endtab %}

{% tab title="JavaScript" %}
javascript
//const client = Client.forTestnet();
//client.setOperator(myAccountId, myPrivateKey);
//-----<enter code
below>-----

//Create new keys
const newAccountPrivateKey = PrivateKey.generateED25519();
const newAccountPublicKey = newAccountPrivateKey.publicKey;

{% endtab %}

{% tab title="Go" %}
go
//client := hedera.ClientForTestnet()
//client.SetOperator(myAccountId, myPrivateKey)
//-----<enter code
below>-----

//Generate new keys for the account you will create
newAccountPrivateKey, err := hedera.PrivateKeyGenerateEd25519()

if err != nil {
    panic(err)
}

newAccountPublicKey := newAccountPrivateKey.PublicKey()

{% endtab %}
{% endtabs %}
```

Step 3: Create a new account

Create a new account using `AccountCreateTransaction()`. Use the public key created in the previous step to enter in the `setKey()` field. This will associate the key pair generated in the previous step with the new account. The public key of the account is visible to the public and can be viewed in a mirror node explorer. The private key is used to authorize account-related transactions like transferring HBAR or tokens from that account to another account. The account

will have an initial balance of 1,000 tinybars funded from your testnet account created by the Hedera portal.

You can view transactions successfully submitted to the network by getting the transaction ID and searching for it in a mirror node explorer. The transaction ID is composed of the account ID that paid for the transaction and the transaction's valid start time e.g. 0.0.1234@1609348302<mark style="color:blue;">.</mark> The transaction's valid start time is the time the transaction begins to be valid on the network. The SDK automatically generates a transaction ID for each transaction behind the scenes.

```
{% tabs %}
{% tab title="Java" %}
java
//Create new account and assign the public key
TransactionResponse newAccount = new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000))
    .execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a new account with 1,000 tinybar starting balance
const newAccount = await new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000))
    .execute(client);

{% endtab %}

{% tab title="Go" %}
go
//Create new account and assign the public key
newAccount, err := hedera.NewAccountCreateTransaction().
    SetKey(newAccountPublicKey).
    SetInitialBalance(hedera.HbarFrom(1000, hedera.HbarUnits.Tinybar)).
    Execute(client)

{% endtab %}
{% endtabs %}
```

Step 4: Get the new account ID

The account ID for the new account is returned in the receipt of the transaction that created the account. The receipt provides information about the transaction like whether it was successful or not and any new entity IDs that were created. Entities include accounts, smart contracts, tokens, files, topics, and scheduled transactions. The account ID is in x.y.z format where z is the account number. The preceding values (x and y) default to zero today and represent the shard and realm number respectively. Your new account ID should result in something like 0.0.1234.

```
{% tabs %}
{% tab title="Java" %}
java
// Get the new account ID
AccountId newAccountId = newAccount.getReceipt(client).accountId;

//Log the account ID
System.out.println("New account ID is: " +newAccountId);

{% endtab %}
```



```

{% tab title="JavaScript" %}
javascript
// Get the new account ID
const getReceipt = await newAccount.getReceipt(client);
const newAccountId = getReceipt.accountId;

//Log the account ID
console.log("The new account ID is: " +newAccountId);

{% endtab %}

{% tab title="Go" %}
go
//Request the receipt of the transaction
receipt, err := newAccount.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the new account ID from the receipt
newAccountId := receipt.AccountID

//Log the account ID
fmt.Printf("The new account ID is %v\n", newAccountId)

{% endtab %}
{% endtabs %}

```

Step 5: Verify the new account balance

Next, you will submit a query to the Hedera test network to return the balance of the new account using the new account ID. The current account balance for the new account should be 1,000 tinybars. Getting the balance of an account is free today.

```

{% tabs %}
{% tab title="Java" %}
java
//Check the new account's balance
AccountBalance accountBalance = new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

System.out.println("New account balance: " +accountBalance.hbars);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Verify the account balance
const accountBalance = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

console.log("The new account balance is: " +accountBalance.hbars.toTinybars() +"
tinybar.");

{% endtab %}

{% tab title="Go" %}
go

```

```
//Create the account balance query
query := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Sign with client operator private key and submit the query to a Hedera network
accountBalance, err := query.Execute(client)
if err != nil {
    panic(err)
}

//Print the balance of tinybars
fmt.Println("The account balance for the new account is ",
accountBalance.Hbars.AsTinybar())

{% endtab %}
{% endtabs %}

{% hint style="success" %}
:star: Congratulations! You have successfully completed the following:

    Created a new Hedera account with an initial balance of 1,000 tinybars.
    Obtained the new account ID by requesting the receipt of the transaction.
    Verified the starting balance of the new account by submitting a query to the
    network.

You are now ready to transfer some HBAR to the new account :money\mouth:!
{% endhint %}
```

Code Check :white\check\mark:

<details>

<summary>Java</summary>

```
java
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.HederaPreCheckStatusException;
import com.hedera.hashgraph.sdk.HederaReceiptStatusException;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.Client;
import com.hedera.hashgraph.sdk.TransactionResponse;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;
import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.AccountBalance;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) throws TimeoutException,
HederaPreCheckStatusException, HederaReceiptStatusException {

        //Grab your Hedera Testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        //Create your Hedera Testnet client
```

```

Client client = Client.forTestnet();
client.setOperator(myAccountId, myPrivateKey);

// Set default max transaction fee & max query payment
client.setDefaultMaxTransactionFee(new Hbar(100));
client.setDefaultMaxQueryPayment(new Hbar(50));

// Generate a new key pair
PrivateKey newAccountPrivateKey = PrivateKey.generateED25519();
PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

//Create new account and assign the public key
TransactionResponse newAccount = new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance( Hbar.fromTinybars(1000))
    .execute(client);

// Get the new account ID
AccountId newAccountId = newAccount.getReceipt(client).accountId;

System.out.println("\nNew account ID: " +newAccountId);

//Check the new account's balance
AccountBalance accountBalance = new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

System.out.println("New account balance: " +accountBalance.hbars);
}
}

```

</details>

<details>

<summary>JavaScript</summary>

```

{% code title="index.js" %}
javascript
const {
  Hbar,
  Client,
  PrivateKey,
  AccountBalanceQuery,
  AccountCreateTransaction,
} = require("@hashgraph/sdk");
require("dotenv").config();

async function environmentSetup() {
  // Grab your Hedera testnet account ID and private key from your .env file
  const myAccountId = process.env.MYACCOUNTID;
  const myPrivateKey = process.env.MYPRIVATEKEY;

  // If we weren't able to grab it, we should throw a new error
  if (myAccountId == null || myPrivateKey == null) {
    throw new Error(
      "Environment variables myAccountId and myPrivateKey must be present"
    );
  }
}

// Create your connection to the Hedera Network
const client = Client.forTestnet();

```

```

client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(100));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(50));

// Create new keys
const newAccountPrivateKey = PrivateKey.generateED25519();
const newAccountPublicKey = newAccountPrivateKey.publicKey;

// Create a new account with 1,000 tinybar starting balance
const newAccount = await new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000))
    .execute(client);

// Get the new account ID
const getReceipt = await newAccount.getReceipt(client);
const newAccountId = getReceipt.accountId;

console.log("\nNew account ID: " + newAccountId);

// Verify the account balance
const accountBalance = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

console.log(
    "The new account balance is: " +
    accountBalance.hbars.toTinybars() +
    " tinybar."
);

return newAccountId;
}
environmentSetup();

{% endcode %}
</details>

<details>

<summary>Go</summary>

go
package main

import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {

    //Loads the .env file and throws an error if it cannot load the variables
    from that file correctly
    err := godotenv.Load(".env")
    if err != nil {

```

```

        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    //Print your testnet account ID and private key to the console to make
sure there was no error
    fmt.Printf("\nThe account ID is = %v\n", myAccountId)
    fmt.Printf("The private key is = %v", myPrivateKey)

    //Create your testnet client
    client := hedera.ClientForTestnet()
    client.SetOperator(myAccountId, myPrivateKey)

    // Set default max transaction fee & max query payment
    client.SetDefaultMaxTransactionFee(hedera.HbarFrom(100,
hedera.HbarUnits.Hbar))
    client.SetDefaultMaxQueryPayment(hedera.HbarFrom(50,
hedera.HbarUnits.Hbar))

    //Generate new keys for the account you will create
    newAccountPrivateKey, err := hedera.PrivateKeyGenerateEd25519()
    if err != nil {
        panic(err)
    }

    newAccountPublicKey := newAccountPrivateKey.PublicKey()

    //Create new account and assign the public key
    newAccount, err := hedera.NewAccountCreateTransaction().
        SetKey(newAccountPublicKey).
        SetInitialBalance(hedera.HbarFrom(1000, hedera.HbarUnits.Tinybar)).
        Execute(client)

    //Request the receipt of the transaction
    receipt, err := newAccount.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    //Get the new account ID from the receipt
    newAccountId := receipt.AccountID

    //Print the new account ID to the console
    fmt.Println("\n")
    fmt.Printf("New account ID: %v\n", newAccountId)

    //Create the account balance query
    query := hedera.NewAccountBalanceQuery().
        SetAccountID(newAccountId)

    //Sign with client operator private key and submit the query to a Hedera
network

```

```

        accountBalance, err := query.Execute(client)
        if err != nil {
            panic(err)
        }

        //Print the balance of tinybars
        fmt.Println("New account balance for the new account is",
accountBalance.Hbars.AsTinybar())
    }

```

</details>

Sample output:

```

bash
New account ID: 0.0.13724748
New account balance: 1000 tinybars.

```

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

environment-set-up.md:

Environment Setup


Summary

This environment setup guide will provide you with the necessary steps to get your development environment ready for building applications on the Hedera Network. You will set up a new project directory, establish a .env environment variable file to store your Hedera Testnet account ID and private keys and configure your Hedera Testnet client.

Prerequisites

Completed the Introduction step.

```

{% hint style="info" %}
Note: You can always check the "Code Check  " section at the bottom of each
page to view the entire code if you run into issues. You can also post your
issue to the respective SDK channel in our Discord community here or on the
GitHub repository here.
{% endhint %}

```

Step 1: Create your project directory

Open your IDE of choice and follow the below steps to create your new project directory.

```

{% tabs %}
{% tab title="Java Gradle" %}
Create a new Gradle project and name it HederaExamples. Add the following
dependencies to your build.gradle file.

```

```

{% code title="build.gradle " %}

```

```

gradle
dependencies {

    implementation 'com.hedera.hashgraph:sdk:2.32.0'
    implementation 'io.grpc:grpc-netty-shaded:1.57.2'
    implementation 'io.github.cdimascio:dotenv-java:2.3.2'
    implementation 'org.slf4j:slf4j-nop:2.0.9'
    implementation 'com.google.code.gson:gson:2.8.8'
}

```

```

{% endcode %}
{% endtab %}

```

```

{% tab title="Java Maven" %}

```

Create a new Maven project and name it HederaExamples. Add the following dependencies to your pom.xml file.

```

{% code title="pom.xml " %}
xml
<dependencies>
    <dependency>
        <groupId>com.hedera.hashgraph</groupId>
        <artifactId>sdk</artifactId>
        <version>2.32.0</version>
    </dependency>
    <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-netty-shaded</artifactId>
        <version>1.57.2</version>
    </dependency>
    <dependency>
        <groupId>io.github.cdimascio</groupId>
        <artifactId>dotenv-java</artifactId>
        <version>2.3.2</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-nop</artifactId>
        <version>2.0.9</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.8</version>
    </dependency>
</dependencies>

```

```

{% endcode %}
{% endtab %}

```

```

{% tab title="JavaScript" %}

```

Open your terminal and create a directory called hello-hedera-js-sdk. After you create the project directory navigate to the directory by running the following command:

```

bash
mkdir hello-hedera-js-sdk && cd hello-hedera-js-sdk

```

Initialize a node.js project in this new directory by running the following command:

```

bash
npm init -y

```

This is what your console should look like after running the command:

```
bash
{
  "name": "hello-hedera-js-sdk",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

Open your terminal and create a project directory called something like hedera-go-examples to store your Go source code.

```
bash
mkdir hedera-go-examples && cd hedera-go-examples
```

```
{% endtab %}
```

```
{% endtabs %}
```

Step 2: Install Dependencies and SDKs

```
{% tabs %}
```

```
{% tab title="Java" %}
```

Create a new Java class and name it something like HederaExamples. Import the following classes to use in your example:

```
java
import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.Client;
import io.github.cdimascio.dotenv.Dotenv;
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.AccountBalance;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.TransferTransaction;
import com.hedera.hashgraph.sdk.TransactionResponse;
import com.hedera.hashgraph.sdk.ReceiptStatusException;
import com.hedera.hashgraph.sdk.PrecheckStatusException;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;

import java.util.concurrent.TimeoutException;
```

Note: You may install the latest version of the Java SDK [here](#).

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

Install the JavaScript SDK with your favorite package manager npm or yarn by running the following command:


```
bash
// Install Hedera's JS SDK with NPM
npm install --save @hashgraph/sdk

// Install with Yarn
yarn add @hashgraph/sdk
```

Install dotenv with your favorite package manager. This will allow our node environment to use your testnet account ID and the private key we will store in a .env file next.

```
bash
// Install with NPM
npm install dotenv

// Install with Yarn
yarn add dotenv
```

Create a index.js file by running the following command:

```
bash
touch index.js
```

Your project structure should look something like this:

```

{% endtab %}
```

```
{% tab title="Go" %}
Create a hederaexamples.go file in hedera-go-examples root directory. You will
write all of your code in this file.
```

```
bash
touch hederaexamples.go
```

Create the Go "module" file by running the below command. The go.mod file defines the module's properties and dependencies and provides a way to manage versioning for Go projects.

```
go
go mod init hederaexamples.go
```

Install the Go SDK:

```
go-module
go get github.com/hashgraph/hedera-sdk-go/v2@latest
```

And the DotEnv package:

```
go-module
go get github.com/joho/godotenv
```

Import the following packages to your hederaexamples.go file:

```
go
package main
```

```
import (
    "fmt"
    "os"

    "github.com/joho/godotenv"
    "github.com/hashgraph/hedera-sdk-go/v2"
)
```

```
{% endtab %}
{% endtabs %}
```

```
{% hint style="info" %}
```

Note: Testnet HBAR is required for this next step. Please follow the instructions to create a Hedera account on the portal before you move on to the next step.

```
{% endhint %}
```

Step 3: Create your .env File

Create the .env file in your project's root directory. The .env file stores your environment variables, such as your account ID and private key.

🔔 Note: If you have not created an account, please do so here before this step.

```
{% tabs %}
```

```
{% tab title="Hedera Developer Portal" %}
```

If you created your testnet account through the developer portal, grab the Hedera Testnet account ID and DER-encoded private key from your Hedera portal profile (see screenshot below) and assign them to the MYACCOUNTID and MYPRIVATEKEY environment variables in your .env file:

```
<figure><figcaption><p>Hedera Developer Portal</p></figcaption></figure>
```

```
markdown
```

```
MYACCOUNTID=0.0.1234
```

```
MYPRIVATEKEY=302e020100300506032b657004220420ed5a93073.....
```

```
{% endtab %}
```

```
{% tab title="Hedera Faucet" %}
```

Alternatively, if you used the faucet to create a testnet account, grab your faucet account ID and the private key (how to export a private key from MetaMask here) and assign them to the MYACCOUNTID and MYPRIVATEKEY environment variables in your .env file:

```
<figure><figcaption></figcaption></figure>
```

```
MYACCOUNTID=0.0.1234
```

```
MYPRIVATEKEY=0x fd154395435c81233b2fc906486f35e068...
```

```
{% endtab %}
```

```
{% endtabs %}
```

Next, you will load your account ID and private key variables from the .env file created in the previous step.

```
{% tabs %}
```

```
{% tab title="Java" %}
```

Within the main method, add your testnet account ID and private key from the

environment file.

```
{% code title="HederaExamples.java" %}
```

```
java
```

```
public class HederaExamples {
```

```
    public static void main(String[] args) {
```

```
        //Grab your Hedera Testnet account ID and private key
```

```
        AccountId myAccountId =
```

```
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
```

```
        PrivateKey myPrivateKey =
```

```
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));
```

```
    }
```

```
}
```

```
{% endcode %}
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
{% code title="index.js" %}
```

```
javascript
```

```
const { Client, PrivateKey, AccountCreateTransaction, AccountBalanceQuery, Hbar,
```

```
TransferTransaction } = require("@hashgraph/sdk");
```

```
require('dotenv').config();
```

```
async function environmentSetup() {
```

```
    //Grab your Hedera testnet account ID and private key from your .env file
```

```
    const myAccountId = process.env.MYACCOUNTID;
```

```
    const myPrivateKey = process.env.MYPRIVATEKEY;
```

```
    // If we weren't able to grab it, we should throw a new error
```

```
    if (!myAccountId || !myPrivateKey) {
```

```
        throw new Error("Environment variables MYACCOUNTID and MYPRIVATEKEY must  
be present");
```

```
    }
```

```
}
```

```
environmentSetup();
```

```
{% endcode %}
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
{% code title="hederaexamples.go" %}
```

```
go
```

```
func main() {
```

```
    //Loads the .env file and throws an error if it cannot load the variables  
from that file correctly
```

```
    err := godotenv.Load(".env")
```

```
    if err != nil {
```

```
        panic(fmt.Errorf("Unable to load environment variables from .env file.  
Error:\n%\n", err))
```

```
    }
```

```
    //Grab your testnet account ID and private key from the .env file
```

```
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
```

```
    if err != nil {
```

```
        panic(err)
```

```
    }
```

```
    myPrivateKey, err := hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
```

```
    if err != nil {
```

```

        panic(err)
    }

    //Print your testnet account ID and private key to the console to make sure
    there was no error
    fmt.Printf("The account ID is = %v\n", myAccountId)
    fmt.Printf("The private key is = %v\n", myPrivateKey)
}

{% endcode %}

```

In your terminal, enter the following command to create your go.mod file. This module is used for tracking dependencies and is required.

```

go-module
go mod init hederaexamples.go

```

Run your code to see your testnet account ID and private key printed to the console.

```

go-module
go run hederaexamples.go

{% endtab %}
{% endtabs %}

```

Step 4: Create your Hedera Testnet client


Create a Hedera Testnet client and set the operator information using the testnet account ID and private key for transaction and query fee authorization. The operator is the default account that will pay for the transaction and query fees in HBAR. You will need to sign the transaction or query with the private key of that account to authorize the payment. In this case, the operator ID is your testnet account ID, and the operator private key is the corresponding testnet account private key.

```

{% hint style="warning" %}
To avoid encountering the INSUFFICIENTTXFEE error while conducting transactions,
you can adjust the maximum transaction fee limit through
the .setDefaultMaxTransactionFee() method. Similarly, the maximum query payment
can be adjusted using the .setDefaultMaxQueryPayment() method.
{% endhint %}

```

<details>

<summary> How to resolve the INSUFFICIENTTXFEE error</summary>

To resolve this error, you must adjust the max transaction fee to a higher value suitable for your needs.

Here is a simple example addition to your code:

```

javascript
const maxTransactionFee = new Hbar(XX); // replace XX with desired fee in Hbar

```

In this example, you can set maxTransactionFee to any value greater than 5 HBAR (or 500,000,000 tinybars) to avoid the "INSUFFICIENT\TX\FEE" error for transactions greater than 5 HBAR. Please replace XX with the desired value.

To implement this new max transaction fee, you use the

setDefaultMaxTransactionFee() method as shown below:

```
javascript
client.setDefaultMaxTransactionFee(maxTransactionFee);
```

</details>

```
{% tabs %}
{% tab title="Java" %}
java
//Create your Hedera Testnet client
Client client = Client.forTestnet();

//Set your account as the client's operator
client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(100));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(50));

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create your Hedera Testnet client
const client = Client.forTestnet();

//Set your account as the client's operator
client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(100));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(50));

{% endtab %}

{% tab title="Go" %}
go
//Create your testnet client
client := hedera.ClientForTestnet()
client.SetOperator(myAccountId, myPrivateKey)

// Set default max transaction fee
client.SetDefaultMaxTransactionFee(hedera.HbarFrom(100, hedera.HbarUnits.Hbar))

// Set max query payment
client.SetDefaultMaxQueryPayment(hedera.HbarFrom(50, hedera.HbarUnits.Hbar))

{% endtab %}
{% endtabs %}
```

Your project environment is now set up to submit transactions and queries to the Hedera test network successfully!

Next, you will learn how to create an account.

Code Check :white\check\mark:

<details>

<summary>Java</summary>

```
<pre class="language-java" data-title="HederaExamples.java"><code class="lang-java">import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.Client;
import io.github.cdimascio.dotenv.Dotenv;
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.AccountBalance;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.TransferTransaction;
import com.hedera.hashgraph.sdk.TransactionResponse;
import com.hedera.hashgraph.sdk.ReceiptStatusException;
import com.hedera.hashgraph.sdk.PrecheckStatusException;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;
import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) {

        //Grab your Hedera Testnet account ID and private key
        AccountId myAccountId =
        AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));+
        PrivateKey myPrivateKey =
        PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));
        //Create your Hedera Testnet client

<strong>        Client client = Client.forTestnet();
</strong>
        client.setOperator(myAccountId, myPrivateKey);
        // Set default max transaction fee &#x26; max query payment
        client.setDefaultMaxTransactionFee(new Hbar(100));
        client.setDefaultMaxQueryPayment(new Hbar(50));

        System.out.println("Client setup complete.");
    }
}
</code></pre>
```

</details>

<details>

<summary>JavaScript</summary>

```
{% code title="index.js" %}
javascript
const {
  Hbar,
  Client,
} = require("@hashgraph/sdk");

require("dotenv").config();

async function environmentSetup() {
  //Grab your Hedera testnet account ID and private key from your .env file
  const myAccountId = process.env.MYACCOUNTID;
  const myPrivateKey = process.env.MYPRIVATEKEY;
```

```

// If we weren't able to grab it, we should throw a new error
if (!myAccountId || !myPrivateKey) {
    throw new Error(
        "Environment variables MYACCOUNTID and MYPRIVATEKEY must be present"
    );
}

//Create your Hedera Testnet client
const client = Client.forTestnet();

//Set your account as the client's operator
client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(100));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(50));

console.log("Client setup complete.");
}
environmentSetup();

{% endcode %}
</details>

<details>

<summary>Go</summary>

{% code title="hederaexamples.go" %}
go
package main

import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {

    //Loads the .env file and throws an error if it cannot load the variables
    from that file correctly
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

```

```

        //Create your testnet client
        client := hedera.ClientForTestnet()
        client.SetOperator(myAccountId, myPrivateKey)
        // Set default max transaction fee & max query payment
        client.SetDefaultMaxTransactionFee(hedera.HbarFrom(100,
hedera.HbarUnits.Hbar))
        client.SetDefaultMaxQueryPayment(hedera.HbarFrom(50,
hedera.HbarUnits.Hbar))

        fmt.Println("Client setup complete.")
    }

{% endcode %}

</details>

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

Contributors: fabianstraubinger99

introduction.md:

Get Your Testnet Account

This guide outlines three methods for creating a Hedera Testnet account: the Developer Portal, the anonymous Faucet, and via HashPack. The Developer Portal requires a sign-up but provides an initial HBAR balance, while the Faucet allows for anonymous account creation and HBAR dispensing with minimal hassle. Choose the method that best suits your preference.

- [➔ Hedera Faucet](#)
- [➔ Hedera Developer Portal](#)
- [➔ ECDSA Account via HashPack](#)

Comparison Table: Hedera Developer Portal vs. Hedera Faucet

This comparison table highlights the differences between using the Hedera Developer Portal and the Hedera Faucet to create testnet accounts and receive HBAR. It helps you decide which path is best for you. The faucet is recommended for EVM developers as it is a familiar tool and only requires a wallet address to auto-create an account.

Feature		Developer Portal	Hedera Faucet
		Account Creation	Sign up/login required, Automatically receive 1000 HBAR upon account creation
	Dispense Limit	Up to 1000 HBAR per day	Up to 100 HBAR per day
	Ease of Use	Centralized dashboard for management	Simple, quick access from link
	Access to Faucet	Available from the portal dashboard	Direct access from the faucet URL
	Additional Features	Personal access tokens (API keys), manage accounts, store private keys	N/A

Hedera Developer Portal

The Hedera developer portal allows you to create a testnet account to receive testnet HBAR. Go to Hedera Developer Portal and follow the instructions to create a testnet account.

```
<figure><figcaption></figcaption></figure>
```

After account creation, your new testnet account will automatically receive 1000 HBAR, and you'll see your account ID and key pair from the portal dashboard (see image below). Copy your account ID and DER-encoded private key for the coding environment setup step.

```
<figure><figcaption></figcaption></figure>
```

The portal dashboard also serves as a central location where you can manage your account IDs and private keys, easily access the testnet HBAR, and unlock features like a personal access token (API key).

```
<figure><figcaption></figcaption></figure>
```

{% hint style="info" %}

Note: Testnet accounts on the developer portal are subject to a daily top-up limit of 1000 HBAR. Accounts do not automatically get topped up. To top up your balance, you must manually request a refill through the portal dashboard every 24 hours.

For clarity, topping up does not add an additional 1000 HBAR to your account balance. Instead, if your account balance falls below this threshold, up to 1000 HBAR is replenished. For example, if your account balance is 500 HBAR, refilling will only add enough HBAR to bring your balance to 1000 HBAR.

{% endhint %}

Hedera Faucet

The Hedera Faucet gives free testnet HBAR anonymously, and you won't need to sign up for an account on the Developer Portal. To use the anonymous faucet, visit the faucet page and enter your EVM wallet address where it says "Enter Wallet Address."

EVM wallet address Hedera account ID

Entering an EVM address initiates an auto account creation flow, creating a new testnet account. Once the testnet account is created successfully, your new account ID will return in the same pop-up window where you requested to receive HBAR. Copy and save your new account ID (see the image below). You will need the account ID AND private key, which will be used later to configure your environment variables.

```
<div>
```

```
<figure><figcaption></figcaption></figure>
```

```
<figure><figcaption></figcaption></figure>
```

</div>

🔔 Please note: This faucet can dispense a maximum of 100 HBAR every 24 hours. If you try to use the faucet before the timer runs out, you'll get the above error message.

<div>

<figure><figcaption></figcaption></figure>

<figure><figcaption></figcaption></figure>

</div>

Testnet ECDSA Account via HashPack

HashPack wallet allows you to create an ECDSA account. Follow the steps below to create an ECDSA testnet account.

1. Download the HashPack wallet extension or mobile app.
2. Open HashPack and follow the prompts to create a new ECDSA account. Select Testnet account and Create a new wallet.
3. Choose the Advanced Creation option and the Seed Phrase (ECDSA) option. Save your seed phrase, private key, and new Hedera account ID in a safe location.
4. Fund your new testnet account by visiting the faucet and following the previous "Hedera Faucet" step.

<div>

<figure><figcaption></figcaption></figure>

<figure><figcaption></figcaption></figure>

</div>

Next Steps

Now that you have a funded Hedera Testnet account and some HBAR, you can start building on Heder. First, set up your environment on the following page!

page-1.md:

Page 1

```
{% hint style="warning" %}
The max transaction fee and max query payment are both set to 100\000\000
tinybar (1 HBAR). This amount can be modified by using
setDefaultMaxTransactionFee()and setMaxQueryPayment().
{% endhint %}
```

Method	Type	Description
<code>Client.<network>.setDefaultRegenerateTransactionId(<regenerateTransactionId>)</code>	boolean	Whether or not to regenerate the transaction IDs
<code>Client.<network>.getDefaultRegenerateTransactionId(<regenerateTransactionId>)</code>	boolean	Get the default regenerate transaction ID
<code>Client.<network>.setDefaultMaxTransactionFee(<fee>)</code>	Hbar	The maximum transaction fee the client is willing to pay
<code>Client.<network>.getDefaultMaxTransactionFee()</code>	Hbar	Get the default max transaction fee that is set
<code>Client.<network>.setMaxQueryPayment(<maxQueryPayment>)</code>	Hbar	The maximum query payment the client will pay.
<code>Client.<network>.getDefaultMaxQueryPayment()</code>	Hbar	Get the default max query payment
<code>Client.<network>.setNetwork(<nodes>)</code>	Map<String, AccountId>	Replace all nodes in this Client with a new set of nodes (e.g. for an Address Book update)
<code>Client.<network>.getNetwork()</code>	Map<String, AccountId>	Get the network nodes
<code>Client.<network>.setRequestTimeout(<requestTimeout>)</code>	Duration	The period of time a transaction or query request will retry from a "busy" network response
<code>Client.<network>.getRequestTimeout()</code>	Duration	Get the period of time a transaction or query request will retry from a "busy" network response
<code>Client.<network>.setMinBackoff(<minBackoff>)</code>	Duration	The minimum amount of time to wait between retries. When retrying, the delay will start at this time and increase exponentially until it reaches the maxBackoff
<code>Client.<network>.getMinBackoff()</code>	Duration	Get the minimum amount of time to wait between retries
<code>Client.<network>.setMaxBackoff(<maxBackoff>)</code>	Duration	The maximum amount of time to wait between retries. Every retry attempt will increase the wait time exponentially until it reaches this time.
<code>Client.<network>.getMaxBackoff()</code>	Duration	Get the maximum amount of time to wait between retries
<code>Client.<network>.setAutoValidateChecksums(<value>)</code>	boolean	Validate checksums
<code>Client.<network>.setCloseTimeout(<closeTimeout>)</code>	Duration	Timeout for closing either a single node when setting a new network, or closing the entire network
<code>Client.<network>.setMaxNodeAttempts(<maxNodeAttempts>)</code>	int	Set the max number of times a node can return a bad gRPC status before we remove it from the list
<code>Client.<network>.getMaxNodeAttempts()</code>	int	Get the max node attempts
<code>Client.<network>.setMinNodeReadmitTime(<minNodeReadmitTime>)</code>	Duration	The min time to wait before attempting to readmit nodes
<code>Client.<network>.getMinNodeReadmitTime()</code>	Duration	Get the minimum node readmit time
<code>Client.<network>.setMaxNodeReadmitTime(<maxNodeReadmitTime>)</code>	Duration	The max time to wait before attempting to readmit nodes
<code>Client.<network>.getMaxNodeReadmitTime()</code>	Duration	Get the max node readmit time

query-data.md:

Query Ledger Data


Summary

In this section, we will guide you through querying your account balance, enabling you to retrieve the most current information about the available funds in your new Hedera account.

Prerequisites [](#pre-requisites)

- Completed the Introduction step.
- Completed the Environment Setup step.
- Completed the Created an Account step.
- Completed the Transfer HBAR step.

{% hint style="info" %}

Note: You can always check the "Code Check  " section at the bottom of each page to view the entire code if you run into issues. You can also post your issue to the respective SDK channel in our Discord community here or on the GitHub repository here.

{% endhint %}

Query the account balance

Get the cost of requesting the query

You can request the cost of a query prior to submitting the query to the Hedera network. Checking an account balance is free of charge today. You can verify that by the method below.

```
{% tabs %}
{% tab title="Java" %}
java
//Request the cost of the query
Hbar queryCost = new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .getCost(client);

System.out.println("The cost of this query is: " +queryCost);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Request the cost of the query
const queryCost = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .getCost(client);

console.log("The cost of query is: " +queryCost);

{% endtab %}

{% tab title="Go" %}
java
//Create the query that you want to submit
balanceQuery := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Get the cost of the query
cost, err := balanceQuery.GetCost(client)
```

```

if err != nil {
    panic(err)
}

println("The account balance query cost is:", cost.String())

{% endtab %}
{% endtabs %}

Get the account balance

You will verify the account balance was updated for the new account by
requesting a get account balance query. The current account balance should be
the sum of the initial balance (1,000 tinybars) plus the transfer amount (1,000
tinybars) and equal to 2,000 tinybars.

{% tabs %}
{% tab title="Java" %}
java
//Check the new account's balance
AccountBalance accountBalanceNew = new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

System.out.println("The account balance after the transfer: "
+accountBalanceNew.hbars);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Check the new account's balance
const getNewBalance = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

console.log("The account balance after the transfer is: "
+getNewBalance.hbars.toTinybars() +" tinybar.")

{% endtab %}

{% tab title="Go" %}
go
//Check the new account's balance
newAccountBalancequery := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Sign with client operator private key and submit the query to a Hedera network
newAccountBalance, err := newAccountBalancequery.Execute(client)
if err != nil {
    panic(err)
}

//Print the balance of tinybars
fmt.Println("The hbar account balance for this account is",
newAccountBalance.Hbars.AsTinybar())

{% endtab %}
{% endtabs %}

{% hint style="success" %}
:star: Congratulations! You have successfully transferred HBAR to another
account on the Hedera Testnet! If you have followed the tutorial from the


```

beginning, you have completed the following thus far:

Set up your Hedera environment to submit transactions and queries.
Created an account.
Transferred HBAR to another account.

Do you want to keep learning? Visit our the SDKs & APIs section to take your learning experience to the next level. You can also find additional Java SDK examples here.

{% endhint %}

Code Check 

Your complete code file should look something like this:

<details>

<summary>Java</summary>

{% code fullWidth="true" %}

```
java
import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.Client;
import io.github.cdimascio.dotenv.Dotenv;
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.AccountBalance;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.TransferTransaction;
import com.hedera.hashgraph.sdk.TransactionResponse;
import com.hedera.hashgraph.sdk.ReceiptStatusException;
import com.hedera.hashgraph.sdk.PrecheckStatusException;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args)
        throws TimeoutException, PrecheckStatusException,
ReceiptStatusException {

        // Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        // Create your connection to the Hedera network
        Client client = Client.forTestnet();

        // Set your account as the client's operator
        client.setOperator(myAccountId, myPrivateKey);

        // Set default max transaction fee & max query payment
        client.setDefaultMaxTransactionFee(new Hbar(100));
        client.setDefaultMaxQueryPayment(new Hbar(50));

        // Generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generateED25519();
        PublicKey newAccountPublicKey =
```

```

newAccountPrivateKey.getPublicKey();

        // Create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
            .setKey(newAccountPublicKey)
            .setInitialBalance(Hbar.fromTinybars(1000))
            .execute(client);

        // Get the new account ID
        AccountId newAccountId =
newAccount.getReceipt(client).accountId;

        System.out.println("\nNew account ID: " + newAccountId);

        // Check the new account's balance
        AccountBalance accountBalance = new AccountBalanceQuery()
            .setAccountId(newAccountId)
            .execute(client);

        System.out.println("New account balance is: " +
accountBalance.hbars);

        // Transfer HBAR
        TransactionResponse sendHbar = new TransferTransaction()
            .addHbarTransfer(myAccountId,
Hbar.fromTinybars(-1000))
            .addHbarTransfer(newAccountId,
Hbar.fromTinybars(1000))
            .execute(client);

        System.out.println("\nThe transfer transaction was: " +
sendHbar.getReceipt(client).status);

        // Request the cost of the query
        Hbar queryCost = new AccountBalanceQuery()
            .setAccountId(newAccountId)
            .getCost(client);

        System.out.println("\nThe cost of this query: " + queryCost);

        // Check the new account's balance
        AccountBalance accountBalanceNew = new AccountBalanceQuery()
            .setAccountId(newAccountId)
            .execute(client);

        System.out.println("The account balance after the transfer: " +
accountBalanceNew.hbars + "\n");
    }
}

```

```
{% endcode %}
```

```
</details>
```

```
<details>
```

```
<summary>JavaScript</summary>
```

```

javascript
const {
  Hbar,
  Client,
  PrivateKey,

```

```

    AccountCreateTransaction,
    AccountBalanceQuery,
    TransferTransaction,
} = require("@hashgraph/sdk")
require("dotenv").config();

async function environmentSetup() {
    // Grab your Hedera testnet account ID and private key from your .env file
    const myAccountId = process.env.MYACCOUNTID;
    const myPrivateKey = process.env.MYPRIVATEKEY;

    // If we weren't able to grab it, we should throw a new error
    if (myAccountId == null || myPrivateKey == null) {
        throw new Error(
            "Environment variables myAccountId and myPrivateKey must be present"
        );
    }

    // Create your connection to the Hedera network
    const client = Client.forTestnet();

    // Set your account as the client's operator
    client.setOperator(myAccountId, myPrivateKey);

    // Set default max transaction fee & max query payment
    client.setDefaultMaxTransactionFee(new Hbar(100));
    client.setDefaultMaxQueryPayment(new Hbar(50));

    // Create new keys
    const newAccountPrivateKey = PrivateKey.generateED25519();
    const newAccountPublicKey = newAccountPrivateKey.publicKey;

    // Create a new account with 1,000 tinybar starting balance
    const newAccountTransactionResponse = await new AccountCreateTransaction()
        .setKey(newAccountPublicKey)
        .setInitialBalance(Hbar.fromTinybars(1000))
        .execute(client);

    // Get the new account ID
    const getReceipt = await newAccountTransactionResponse.getReceipt(client);
    const newAccountId = getReceipt.accountId;

    console.log("\nNew account ID: " + newAccountId);

    // Verify the account balance
    const accountBalance = await new AccountBalanceQuery()
        .setAccountId(newAccountId)
        .execute(client);

    console.log(
        "New account balance is: " +
        accountBalance.hbars.toTinybars() +
        " tinybars."
    );

    // Create the transfer transaction
    const sendHbar = await new TransferTransaction()
        .addHbarTransfer(myAccountId, Hbar.fromTinybars(-1000))
        .addHbarTransfer(newAccountId, Hbar.fromTinybars(1000))
        .execute(client);

    // Verify the transaction reached consensus
    const transactionReceipt = await sendHbar.getReceipt(client);

```



```

console.log(
    "The transfer transaction from my account to the new account was: " +
    transactionReceipt.status.toString()
);

// Request the cost of the query
const queryCost = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .getCost(client);

console.log("\nThe cost of query is: " + queryCost);

// Check the new account's balance
const getNewBalance = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

console.log(
    "The account balance after the transfer is: " +
    getNewBalance.hbars.toTinybars() +
    " tinybars."
);
}
environmentSetup();

```

</details>

<details>

<summary>Go</summary>

```

go
package main

import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {
    //Loads the .env file and throws an error if it cannot load the variables
    from that file correctly
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%\v\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }
}

```

```

//Create your testnet client
client := hedera.ClientForTestnet()
client.SetOperator(myAccountId, myPrivateKey)

// Set default max transaction fee & max query payment
client.SetDefaultMaxTransactionFee(hedera.HbarFrom(100,
hedera.HbarUnits.Hbar))
client.SetDefaultMaxQueryPayment(hedera.HbarFrom(50,
hedera.HbarUnits.Hbar))

//Generate new keys for the account you will create
newAccountPrivateKey, err := hedera.PrivateKeyGenerateEd25519()
if err != nil {
    panic(err)
}

newAccountPublicKey := newAccountPrivateKey.PublicKey()

//Create new account and assign the public key
newAccount, err := hedera.NewAccountCreateTransaction().
    SetKey(newAccountPublicKey).
    SetInitialBalance(hedera.HbarFrom(1000, hedera.HbarUnits.Tinybar)).
    Execute(client)

//Request the receipt of the transaction
receipt, err := newAccount.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the new account ID from the receipt
newAccountId := receipt.AccountID

//Print the new account ID to the console
fmt.Println("\n")
fmt.Printf("New account ID: %v\n", newAccountId)

//Create the account balance query
query := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Sign with client operator private key and submit the query to a Hedera
network
accountBalance, err := query.Execute(client)
if err != nil {
    panic(err)
}

//Print the balance of tinybars
fmt.Println("New account balance for the new account is",
accountBalance.Hbars.AsTinybar())

//Transfer hbar from your testnet account to the new account
transaction := hedera.NewTransferTransaction().
    AddHbarTransfer(myAccountId, hedera.HbarFrom(-1000,
hedera.HbarUnits.Tinybar)).
    AddHbarTransfer(newAccountId, hedera.HbarFrom(1000,
hedera.HbarUnits.Tinybar))

// Submit the transaction to a Hedera network
txResponse, err := transaction.Execute(client)

if err != nil {

```

```

        panic(err)
    }

    // Request the receipt of the transaction
    transferReceipt, err := txResponse.GetReceipt(client)

    if err != nil {
        panic(err)
    }

    // Get the transaction consensus status
    transactionStatus := transferReceipt.Status

    fmt.Printf("\nThe transaction consensus status is %v\n\n",
transactionStatus)

    //Create the query that you want to submit
    balanceQuery := hedera.NewAccountBalanceQuery().
        SetAccountID(newAccountId)

    //Get the cost of the query
    cost, err := balanceQuery.GetCost(client)

    if err != nil {
        panic(err)
    }

    fmt.Println("The account balance query cost is:", cost.String())

    //Check the new account's balance
    newAccountBalanceQuery := hedera.NewAccountBalanceQuery().
        SetAccountID(newAccountId)

    //Sign with client operator private key and submit the query to a Hedera
network
    newAccountBalance, err := newAccountBalanceQuery.Execute(client)
    if err != nil {
        panic(err)
    }

    //Print the balance of tinybars
    fmt.Println("The HBAR balance for this account is",
newAccountBalance.Hbars.AsTinybar())
}

```

</details>

Sample output:

```

bash
New account ID: 0.0.13724748
New account balance: 1000 tinybars.

```

The transfer transaction from my account to the new account was: SUCCESS

The cost of query: 0 tñ

The account balance after the transfer: 2000 tinybars.

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

README.md:

cover: >-

../.gitbook/assets/Hero-Desktop-EnterpriseApplications2022-12-08-192047ivzd.webp

coverY: 24.9050380401673

layout:

cover:

visible: true

size: full

title:

visible: true

description:

visible: false

tableOfContents:

visible: true

outline:

visible: true

pagination:

visible: true

Getting Started

transfer-hbar.md:

Transfer HBAR

Summary

In this section, you will learn how to transfer HBAR from your account to another on the Hedera test network.


Prerequisites

Completed the Introduction step.

Completed the Environment Setup step.

Completed the Created an Account step.

{% hint style="info" %}

Note: You can always check the "Code Check  " section at the bottom of each page to view the entire code if you run into issues. You can also post your issue to the respective SDK channel in our Discord community here or on the GitHub repository here.

{% endhint %}

Step 1. Create a transfer transaction

Use your new account created in the "Create an account" section and transfer 1,000 tinybars from your account to the new account. The account sending the HBAR needs to sign the transaction using its private keys to authorize the transfer. Since you are transferring from the account associated with the client, you do not need to explicitly sign the transaction as the operator account(account transferring the HBAR) signs all transactions to authorize the payment of the transaction fee.

```

{% tabs %}
{% tab title="Java" %}
java
//System.out.println("The new account balance is: " +accountBalance.hbars);
//-----<enter code
below>-----

//Transfer HBAR
TransactionResponse sendHbar = new TransferTransaction()
    .addHbarTransfer(myAccountId, Hbar.fromTinybars(-1000)) //Sending account
    .addHbarTransfer(newAccountId, evmAddress, Hbar.fromTinybars(1000))
//Receiving account
    .execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//console.log("The new account balance is: " +accountBalance.hbars.toTinybars()
+" tinybar.");
//-----<enter code
below>-----

//Create the transfer transaction
const sendHbar = await new TransferTransaction()
    .addHbarTransfer(myAccountId, Hbar.fromTinybars(-1000)) //Sending account
    .addHbarTransfer(newAccountId, evmAddress, Hbar.fromTinybars(1000))
//Receiving account
    .execute(client);

{% endtab %}

{% tab title="Go" %}
java
//Print the balance of tinybars
//fmt.Println("The account balance for the new account is ",
accountBalance.Hbars.AsTinybar())
//-----<enter code
below>-----

//Transfer hbar from your testnet account to the new account
transaction := hedera.NewTransferTransaction().
    AddHbarTransfer(myAccountId, hedera.HbarFrom(-1000,
hedera.HbarUnits.Tinybar)).
    AddHbarTransfer(newAccountId, evmAddress, hedera.HbarFrom(1000,
hedera.HbarUnits.Tinybar))

//Submit the transaction to a Hedera network
txResponse, err := transaction.Execute(client)

if err != nil {
    panic(err)
}

{% endtab %}
{% endtabs %}

{% hint style="info" %}
Note: The net value of the transfer must equal zero (the total number of HBAR
sent by the sender must equal the total number of HBAR received by the
recipient).
{% endhint %}

```

Step 2. Verify the transfer transaction reached consensus

To verify the transfer transaction reached consensus by the network, you will submit a request to obtain the receipt of the transaction. The receipt status will let you know if the transaction was successful (reached consensus) or not.

```
{% tabs %}
{% tab title="Java" %}
java
System.out.println("The transfer transaction was: "
+sendHbar.getReceipt(client).status);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Verify the transaction reached consensus
const transactionReceipt = await sendHbar.getReceipt(client);
console.log("The transfer transaction from my account to the new account was: "
+ transactionReceipt.status.toString());

{% endtab %}


{% tab title="Go" %}
java
//Request the receipt of the transaction
transferReceipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := transferReceipt.Status

fmt.Printf("The transaction consensus status is %v\n", transactionStatus)

{% endtab %}
{% endtabs %}
```

Code Check 

Your complete code file should look something like this:

<details>

<summary>Java</summary>

```
{% code title="HederaExamples.java" %}
java
import com.hedera.hashgraph.sdk.Hbar;
import com.hedera.hashgraph.sdk.Client;
import io.github.cdimascio.dotenv.Dotenv;
import com.hedera.hashgraph.sdk.AccountId;
import com.hedera.hashgraph.sdk.PublicKey;
import com.hedera.hashgraph.sdk.PrivateKey;
import com.hedera.hashgraph.sdk.AccountBalance;
import com.hedera.hashgraph.sdk.AccountBalanceQuery;
import com.hedera.hashgraph.sdk.TransferTransaction;
import com.hedera.hashgraph.sdk.TransactionResponse;
```

```

import com.hedera.hashgraph.sdk.ReceiptStatusException;
import com.hedera.hashgraph.sdk.PrecheckStatusException;
import com.hedera.hashgraph.sdk.AccountCreateTransaction;

import java.util.concurrent.TimeoutException;

public class HederaExamples {

    public static void main(String[] args) throws TimeoutException,
HederaPreCheckStatusException, HederaReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        // Create your connection to the Hedera network
        const client = Client.forTestnet();

        //Set your account as the client's operator
        client.setOperator(myAccountId, myPrivateKey);

        // Set default max transaction fee & max query payment
        client.setDefaultMaxTransactionFee(new Hbar(100));
        client.setDefaultMaxQueryPayment(new Hbar(50));

        // Generate a new key pair
        PrivateKey newAccountPrivateKey = PrivateKey.generateED25519();
        PublicKey newAccountPublicKey = newAccountPrivateKey.getPublicKey();

        //Create new account and assign the public key
        TransactionResponse newAccount = new AccountCreateTransaction()
            .setKey(newAccountPublicKey)
            .setInitialBalance( Hbar.fromTinybars(1000))
            .execute(client);

        // Get the new account ID
        AccountId newAccountId = newAccount.getReceipt(client).accountId;

        System.out.println("\nNew account ID: " +newAccountId);

        //Check the new account's balance
        AccountBalance accountBalance = new AccountBalanceQuery()
            .setAccountId(newAccountId)
            .execute(client);

        //Transfer HBAR
        TransactionResponse sendHbar = new TransferTransaction()
            .addHbarTransfer(myAccountId, Hbar.fromTinybars(-1000))
            .addHbarTransfer(newAccountId, Hbar.fromTinybars(1000))
            .execute(client);

        System.out.println("The transfer transaction was: "
+sendHbar.getReceipt(client).status);

    }
}

{% encode %}

</details>

<details>

```

<summary>JavaScript</summary>

```
{% code title="index.js" %}
javascript
const {
  Hbar,
  Client,
  PrivateKey,
  AccountBalanceQuery,
  TransferTransaction,
  AccountCreateTransaction,
} = require("@hashgraph/sdk");
require("dotenv").config();

async function environmentSetup() {
  // Grab your Hedera testnet account ID and private key from your .env file
  const myAccountId = process.env.MYACCOUNTID;
  const myPrivateKey = process.env.MYPRIVATEKEY;

  //

  // If we weren't able to grab it, we should throw a new error
  if (myAccountId == null || myPrivateKey == null) {
    throw new Error(
      "Environment variables myAccountId and myPrivateKey must be present"
    );
  }

  // Create your connection to the Hedera network
  const client = Client.forTestnet();

  //Set your account as the client's operator
  client.setOperator(myAccountId, myPrivateKey);

  // Set default max transaction fee & max query payment
  client.setDefaultMaxTransactionFee(new Hbar(100));
  client.setDefaultMaxQueryPayment(new Hbar(50));

  // Create new keys
  const newAccountPrivateKey = PrivateKey.generateED25519();
  const newAccountPublicKey = newAccountPrivateKey.publicKey;

  // Create a new account with 1,000 tinybar starting balance
  const newAccountTransactionResponse = await new AccountCreateTransaction()
    .setKey(newAccountPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000))
    .execute(client);

  // Get the new account ID
  const getReceipt = await newAccountTransactionResponse.getReceipt(client);
  const newAccountId = getReceipt.accountId;

  console.log("\nNew account ID: " + newAccountId);

  // Verify the account balance
  const accountBalance = await new AccountBalanceQuery()
    .setAccountId(newAccountId)
    .execute(client);

  console.log(
    "\nNew account balance is: " +
    accountBalance.hbars.toTinybars() +
    " tinybars."
  )
}
```



```

);

// Create the transfer transaction
const sendHbar = await new TransferTransaction()
    .addHbarTransfer(myAccountId, Hbar.fromTinybars(-1000))
    .addHbarTransfer(newAccountId, Hbar.fromTinybars(1000))
    .execute(client);

// Verify the transaction reached consensus
const transactionReceipt = await sendHbar.getReceipt(client);
console.log(
    "\nThe transfer transaction from my account to the new account was: " +
    transactionReceipt.status.toString()
);
}
environmentSetup();

{% endcode %}
</details>

<details>

<summary>Go</summary>

go
package main

import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {

    //Loads the .env file and throws an error if it cannot load the variables
    from that file correctly
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%\v\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    //Create your testnet client
    client := hedera.ClientForTestnet()
    client.SetOperator(myAccountId, myPrivateKey)

    // Set default max transaction fee & max query payment
    client.SetDefaultMaxTransactionFee(hedera.HbarFrom(100,

```

```

hedera.HbarUnits.Hbar))
    client.SetDefaultMaxQueryPayment(hedera.HbarFrom(50,
hedera.HbarUnits.Hbar))

    //Generate new keys for the account you will create
    newAccountPrivateKey, err := hedera.PrivateKeyGenerateEd25519()
    if err != nil {
        panic(err)
    }

    newAccountPublicKey := newAccountPrivateKey.PublicKey()

    //Create new account and assign the public key
    newAccount, err := hedera.NewAccountCreateTransaction().
        SetKey(newAccountPublicKey).
        SetInitialBalance(hedera.HbarFrom(1000, hedera.HbarUnits.Tinybar)).
        Execute(client)

    //Request the receipt of the transaction
    receipt, err := newAccount.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    //Get the new account ID from the receipt
    newAccountId := receipt.AccountID

    //Print the new account ID to the console
    fmt.Printf("The new account ID is %v\n", newAccountId)

    //Create the account balance query
    query := hedera.NewAccountBalanceQuery().
        SetAccountID(newAccountId)

    //Sign with client operator private key and submit the query to a Hedera
network
    accountBalance, err := query.Execute(client)
    if err != nil {
        panic(err)
    }

    //Print the balance of tinybars
    fmt.Println("The account balance for the new account is",
accountBalance.Hbars.AsTinybar())

    //Transfer hbar from your testnet account to the new account
    transaction := hedera.NewTransferTransaction().
        AddHbarTransfer(myAccountId, hedera.HbarFrom(-1000,
hedera.HbarUnits.Tinybar)).
        AddHbarTransfer(newAccountId, hedera.HbarFrom(1000,
hedera.HbarUnits.Tinybar))

    //Submit the transaction to a Hedera network
    txResponse, err := transaction.Execute(client)

    if err != nil {
        panic(err)
    }

    //Request the receipt of the transaction
    transferReceipt, err := txResponse.GetReceipt(client)

    if err != nil {
        panic(err)
    }

```

```

    }

    //Get the transaction consensus status
    transactionStatus := transferReceipt.Status

    fmt.Printf("The transaction consensus status is %v\n", transactionStatus)
}

```

</details>

Sample output:

```

bash
New account ID: 0.0.4382765

New account balance is: 1000 tinybars.

The transfer transaction from my account to the new account was: SUCCESS

```

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

schedule-your-first-transaction.md:

Schedule Your First Transaction

Summary

In this tutorial, you'll learn how to create and sign a scheduled transaction. Scheduled Transactions enable multiple parties to easily, inexpensively, and natively schedule and execute any type of Hedera transaction together. Once a transaction is scheduled, additional signatures can be submitted via a ScheduleSign transaction. After the last signature is received within the allotted timeframe, the scheduled transaction will execute.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

Table of Contents

1. Create Transaction
2. Schedule Transaction
3. Submit Signatures
4. Verify Schedule was Triggered
5. Verify Scheduled Transaction Executed

1. Create a transaction to schedule

First, you will need to build the transaction to schedule. In the example below, you will create a transfer transaction. The sender account has a threshold key structure that requires 2 out of the 3 keys to sign the transaction to authorize the transfer amount.

```
{% tabs %}
{% tab title="Java" %}
java
//Create a transaction to schedule
TransferTransaction transaction = new TransferTransaction()
    .addHbarTransfer(senderAccount, Hbar.fromTinybars(-1))
    .addHbarTransfer(recipientAccount, Hbar.fromTinybars(1));

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a transaction to schedule
const transaction = new TransferTransaction()
    .addHbarTransfer(senderAccount, Hbar.fromTinybars(-1))
    .addHbarTransfer(recipientAccount, Hbar.fromTinybars(1));

{% endtab %}

{% tab title="Go" %}
go
//Create a transaction to schedule
transaction := hedera.NewTransferTransaction().
    AddHbarTransfer(senderAccount, hedera.HbarFromTinybar(-1)).
    AddHbarTransfer(recipientAccount, hedera.HbarFromTinybar(1))

{% endtab %}
{% endtabs %}
```

2. Schedule the transfer transaction

Next, you will schedule the transfer transaction by submitting a `ScheduleCreate` transaction to the network. Once the transfer transaction is scheduled, you can obtain the schedule ID from the receipt of the `ScheduleCreate` transaction. The schedule ID identifies the schedule that scheduled the transfer transaction. The schedule ID can be shared with the three signatories. The schedule is immutable unless the admin key is specified during creation.

The scheduled transaction ID of the transfer transaction can also be returned from the receipt of the `ScheduleCreate` transaction. You will notice that the transaction ID for a scheduled transaction includes a `?scheduled` flag e.g. `0.0.9401@1620177544.531971543?scheduled`. All transactions that have been scheduled will include this flag.

You can optionally add signatures you may have during the creation of the `ScheduleCreate` transaction by calling `.freezeWith(client)` and `.sign()` methods. This might make sense if you are one of the required signatures for the scheduled transaction.

Visit the page below to view additional properties that can be set when building a `ScheduleCreate` transaction.

```
{% content-ref url="../../../sdks-and-apis/sdks/schedule-transaction/create-a-schedule-transaction.md" %}
create-a-schedule-transaction.md
{% endcontent-ref %}
```

```

{% tabs %}
{% tab title="Java" %}
java
//Schedule a transaction
TransactionResponse scheduleTransaction = new ScheduleCreateTransaction()
    .setScheduledTransaction(transaction)
    .execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = scheduleTransaction.getReceipt(client);

//Get the schedule ID
ScheduleId scheduleId = receipt.scheduleId;
System.out.println("The schedule ID is " +scheduleId);

//Get the scheduled transaction ID
TransactionId scheduledTxId = receipt.scheduledTransactionId;
System.out.println("The scheduled transaction ID is " +scheduledTxId);

{% endtab %}

{% tab title="JavaScript" %}
<pre class="language-javascript"><code class="lang-javascript">//Schedule a
transaction
<strong>const scheduleTransaction = await new ScheduleCreateTransaction()
</strong>    .setScheduledTransaction(transaction)
    .execute(client);

//Get the receipt of the transaction
const receipt = await scheduleTransaction.getReceipt(client);

//Get the schedule ID
const scheduleId = receipt.scheduleId;
console.log("The schedule ID is " +scheduleId);

//Get the scheduled transaction ID
const scheduledTxId = receipt.scheduledTransactionId;
console.log("The scheduled transaction ID is " +scheduledTxId);
</code></pre>
{% endtab %}

{% tab title="Go" %}
go
//Schedule a transaction
scheduleTransaction, err := hedera.NewScheduleCreateTransaction().
    SetScheduledTransaction(transaction)
if err != nil {
    panic(err)
}

submitScheduleTx, err := scheduleTransaction.Execute(client)
if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err := submitScheduleTx.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the schedule ID
scheduleId := receipt.ScheduleID

```

```

fmt.Printf("The schedule ID %v\n", scheduleId)

//Get the scheduled transaction ID
scheduleTxId := receipt1.ScheduledTransactionID
fmt.Printf("The scheduled transaction ID is %v\n", scheduleTxId)

{% endtab %}
{% endtabs %}

```

3. Submit Signatures

Submit one of the required signatures for the transfer transaction

The signatures are submitted to the network via a ScheduleSign transaction. The ScheduleSign transaction requires the schedule ID of the schedule and the signature of one or more of the required keys. The scheduled transaction has 30 minutes from the time it is scheduled to receive all of its signatures; if the signature requirements are not met, the scheduled transaction will expire.

In the example below, you will submit one signature, confirm the transaction was successful, and get the schedule info to verify the signature was added to the schedule. To verify the signature was added, you can compare the public key of the submitted signature to the public key that is returned from the schedule info request.

```

{% tabs %}
{% tab title="Java" %}
java
//Submit the first signatures
TransactionResponse signature1 = new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey1)
    .execute(client);

//Verify the transaction was successful and submit a schedule info request
TransactionReceipt receipt1 = signature1.getReceipt(client);
System.out.println("The transaction status is " +receipt1.status);

ScheduleInfo query1 = new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);

//Confirm the signature was added to the schedule
System.out.println(query1);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Submit the first signature
const signature1 = await (await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey1))
    .execute(client);

//Verify the transaction was successful and submit a schedule info request
const receipt1 = await signature1.getReceipt(client);
console.log("The transaction status is " +receipt1.status.toString());

const query1 = await new ScheduleInfoQuery()

```

```

        .setScheduleId(scheduleId)
        .execute(client);

//Confirm the signature was added to the schedule
console.log(query1);

{% endtab %}

{% tab title="Go" %}
go
//Submit the first signature
signature1, err := hedera.NewScheduleSignTransaction().
    SetScheduleID(scheduleId).
    FreezeWith(client)
if err != nil {
    panic(err)
}

//Verify the transaction was successful and submit a schedule info request
submitTx, err := signature1.Sign(signerKey1).Execute(client)
if err != nil {
    panic(err)
}

receipt1, err := submitTx.GetReceipt(client)
if err != nil {
    panic(err)
}

status := receipt1.Status

fmt.Printf("The transaction status is %v\n", status)

query1, err := hedera.NewScheduleInfoQuery().
    SetScheduleID(scheduleId).
    Execute(client)

//Confirm the signature was added to the schedule
fmt.Print(query1)

{% endtab %}
{% endtabs %}

```

Submit the second signature

Next, you will submit the second signature and verify the transaction was successful by requesting the receipt. For example purposes, you have access to all three signing keys. But the idea here is that each signer can independently submit their signature to the network.

```

{% tabs %}
{% tab title="Java" %}
{% code title="Java" %}
java
//Submit the second signature
TransactionResponse signature2 = new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey2)
    .execute(client);

//Verify the transaction was successful
TransactionReceipt receipt2 = signature2.getReceipt(client);
System.out.println("The transaction status" +receipt2.status);

```

```

{% endcode %}
{% endtab %}

{% tab title="JavaScript" %}
javascript
//Submit the second signature
const signature2 = await (await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey2))
    .execute(client);

//Verify the transaction was successful
const receipt2 = await signature2.getReceipt(client);
console.log("The transaction status " +receipt2.status.toString());

{% endtab %}

{% tab title="Go" %}
go
//Submit the second signature
signature2, err := hedera.NewScheduleSignTransaction().
    SetScheduleID(scheduleId).
    FreezeWith(client)
if err != nil {
    panic(err)
}

//Verify the transaction was successful and submit a schedule info request
submitTx2, err := signature2.Sign(signerKey2).Execute(client)
if err != nil {
    panic(err)
}

receipt2, err := submitTx2.GetReceipt(client)
if err != nil {
    panic(err)
}

status2 := receipt2.Status

fmt.Printf("The transaction status is %v\n", status2)

{% endtab %}
{% endtabs %}

```

4. Verify the schedule was triggered

The schedule is triggered after it meets its minimum signing requirements. As soon as the last required signature is submitted, the schedule executes the scheduled transaction. To verify the schedule was triggered, query for the schedule info. When the schedule info is returned, you should notice both public keys that signed in the signatories field and the timestamp recorded for when the schedule transaction was executed in the executedAt field.

```

{% tabs %}
{% tab title="Java" %}
java
//Get the schedule info
ScheduleInfo query2 = new ScheduleInfoQuery()
    .setScheduleId(scheduleId)

```



```

        .execute(client);

System.out.println(query2);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Get the schedule info
const query2 = await new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);

console.log(query2);

{% endtab %}

{% tab title="Go" %}
go
//Get the schedule info
query2, err := hedera.NewScheduleInfoQuery().
    SetScheduleID(scheduleId).
    Execute(client)

fmt.Print(query2)

{% endtab %}
{% endtabs %}

```

5. Verify the scheduled transaction executed

When the scheduled transaction (transfer transaction) executes a record is produced that contains the transaction details. The scheduled transaction record can be requested immediately after the transaction has executed and includes the corresponding schedule ID. If you do not know when the scheduled transaction will execute, you can always query a mirror node using the scheduled transaction ID without the ?scheduled flag to get a copy of the transaction record.

```

{% tabs %}
{% tab title="Java" %}
java
//Get the scheduled transaction record
TransactionRecord scheduledTxRecord =
TransactionId.fromString(scheduledTxId.toString()).getRecord(client);
System.out.println("The scheduled transaction record is: " +scheduledTxRecord);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Get the scheduled transaction record
const scheduledTxRecord = await
TransactionId.fromString(scheduledTxId.toString()).getRecord(client);
console.log("The scheduled transaction record is: " +scheduledTxRecord);

{% endtab %}

{% tab title="Go" %}
go
scheduledTxRecord, err := scheduledTransactionId.GetRecord(client)
fmt.Printf("The scheduled transaction record is %v\n", scheduledTxRecord)

```

```
{% endtab %}
{% endtabs %}

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}
```

community-mirror-nodes.md:

```
---
description: Network explorers hosted by the community
cover: >-
  ../.gitbook/assets/Hero-Desktop-NetworkExplorers2022-12-07-020704ehza
  (1).webp
coverY: -209.48275862068965
---
```

Network Explorers and Tools

Hedera network explorers are tools for tracking activity on the Hedera network. Mirror nodes provide real-time data on transactions, while network explorers offer a user-friendly interface for viewing and searching those transactions.

Check out some of the community-hosted network explorer services listed below. Each community-hosted network explorer may have their own unique features and experience.

https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard	https://explorer.arkhia.io/#/mainnet/dashboard
https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home	https://app.dragonglass.me/hedera/home
https://hashscan.io/	https://hashscan.io/	https://hashscan.io/	https://hashscan.io/	https://hashscan.io/	https://hashscan.io/	https://hashscan.io/
https://hederaexplorer.io/	https://hederaexplorer.io/	https://hederaexplorer.io/	https://hederaexplorer.io/	https://hederaexplorer.io/	https://hederaexplorer.io/	https://hederaexplorer.io/
https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/
https://www.lworks.io/	https://www.lworks.io/	https://www.lworks.io/	https://www.lworks.io/	https://www.lworks.io/	https://www.lworks.io/	https://www.lworks.io/
https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/	https://explore.lworks.io/

FAQ

<details>

<summary><mark style="color:blue;">How do I search for a transaction?</mark></summary>

To search for a specific transaction, you can use the unique transaction ID.

The transaction ID should look something like this:

0.0.48750443@1671560120.085845879

</details>

<details>

<summary><mark style="color:blue;">How do I get the transaction ID?</mark></summary>

The transaction ID can be automatically generated by the SDK, manually created and associated with a transaction, or obtained from the receipt or record after the transaction has been processed. It serves as a unique identifier for the transaction and can be used to search for and view its details.

</details>

<details>

<summary><mark style="color:blue;">How do I search for an entity (account, topic, tokens, smart contracts)?</mark></summary>

You can search by the unique ID of the entity you are looking for. The entity ID format is 0.0.entityNumber.

For example, 0.0.2 is an account ID and you search for that account using that ID.

</details>

<details>

<summary><mark style="color:blue;">How do I get the entity ID?</mark></summary>

Entity IDs are returned in the receipt of the transaction that created them. Entities include accounts, topics, smart contracts, schedules, and tokens.\

For example, if you create a new account using the AccountCreateTransaction in the SDK, you can get the new account ID from the transaction receipt.

</details>

<details>

<summary><mark style="color:blue;">Can I host my own Hedera network explorer?</mark></summary>

Yes, you can! You can create your own custom Hedera network explorer using the Mirror Node REST APIs or take a look at the Hedera Mirror Node Explorer open-source project.

</details>

<details>

<summary><mark style="color:blue;">How can I add a network explorer to this page?</mark></summary>

To add a network explorer to this page, refer to the contributing guide and open an issue in the hedera-docs repository. Please include the following information within the issue:

Network explorer name
Link to network explorer
High-resolution logo

</details>

README.md:

cover: ../../gitbook/assets/Hero-Desktop-NetworkExplorers2022-12-07-020704ehza.webp
coverY: -391

Networks

multinode-configuration.md:

Multinode Configuration

Using Multinode Configuration

Multinode configuration is an advanced feature designed for specific scenarios that require multiple consensus nodes. This configuration demands higher resources and involves more complexity, making it suitable primarily for testing and development environments. Before attempting to use the multinode setup, it's crucial to ensure that the local node operates correctly in the default single-node mode.

<details>

<summary>Multinode Mode Requirements</summary>

To run the multinode mode, ensure the following configurations are set at minimum in Docker Settings -> Resources and at least 14 GB of memory are available for Docker:

CPUs: 6
Memory: 14 GB
Swap: 1 GB
Disk Image Size: 64 GB

</details>

{% hint style="info" %}

🔔 Note: Creating a decentralized network where each node runs independently on its own machine is currently unsupported. Nonetheless, advanced networking and configuration capabilities are available, allowing nodes to communicate with each other similar to their interactions on the Hedera Mainnet.

```
{% endhint %}
```

Starting Multinode Mode

To start Hedera Local Node in multinode mode, append the `--multinode` flag to your start command. For example:

bash

npm command to start the local network in multinode mode

```
npm run start -- -d --multinode
```

docker command to start the local network in multinode mode

```
docker compose up -d --multinode
```

Verify the successful launch of multinode mode by inspecting Docker output of `docker ps --format "table {{.Names}}" | grep network` or the Docker Desktop dashboard. You should identify four running nodes:


bash

```
network-node
```

```
network-node-1
```

```
network-node-2
```

```
network-node-3
```

 **Note:** In multinode mode, you need at least three healthy nodes for operational network.

Starting and Stopping Nodes

Individual nodes can be started or stopped to test consensus, synchronization, and node selection processes using npm or docker management commands:

<details>

<summary>npm commands</summary>

bash

npm command to start an individual node

```
npm run start network-node-3
```

npm command to stop an individual node

```
npm run stop network-node-3
```

npm command to restart an individual node

```
npm run restart network-node-3
```

</details>

<details>

<summary>docker commands</summary>

bash

Docker command to start an individual node

```
docker compose start network-node-3
```

Docker command to stop an individual node

```
docker compose stop network-node-3
```

Docker command to restart an individual node

```
docker compose restart network-node-3
```

Docker command to check logs of the individual node
docker compose logs network-node-3 -f

Docker command to stop local network and remove containers
docker compose down

</details>

Alternatively, run `docker compose down -v; git clean -xfd; git reset --hard` to stop the local node and reset it to its original state.

Multinode Mode Diagram

The following diagram illustrates the architecture and flow of data in multinode mode.

<figure><figcaption><p>Multinode mode diagram</p></figcaption></figure>

README.md:

Localnet

Introduction

The Hedera Localnet provides developers with a comprehensive framework for locally testing and refining Hedera-based applications. By operating outside the public networks, Localnet is crucial in the software development lifecycle, eliminating network I/O bottlenecks, minimizing shared resource conflicts, and offering complete control over network configurations. This local network comprises of two primary product offerings, Local Node and Solo, each serving distinct purposes in the development and testing process. For developers just getting started with Local Node, here is the recommended path for testing:

<figure><figcaption><p>Hedera Local Node Testing Workflow</p></figcaption></figure>

1. Local Node (single & multinode mode): Begin by testing your initial prototype on a local node. This step allows for quick iterations and debugging in a controlled environment. If your app needs to handle more complex scenarios, run the Multinode configuration.
2. Solo: Then move to Solo for advanced testing under realistic network conditions.
3. Previewnet: Then test on Hedera Previewnet for latest/bleeding edge/upcoming code verification.
4. Testnet: Lastly, testing on the test network for stable code verification before deploying it on Hedera Mainnet.

Local Node

Overview

The Local Node replicates a Hedera network comprised of a single node (or few if configured) on a developer's local machine, offering a controlled environment for developing, testing, and experimenting with decentralized applications (dApps). This local setup utilizes Docker Compose to create multiple containers, each with a specific role within the network, including but not limited to:

Consensus Node: Simulates the behavior of Hedera's consensus mechanism, processing/ordering transactions and participating in the network's consensus algorithm.

Mirror Node: Provides access to the historical data, transaction records, and the current state of the network without participating in consensus. This is useful for querying and analytics.

JSON-RPC Relay: Offers a local JSON-RPC implementation of the Ethereum JSON-RPC APIs for Hedera to enable interactions with smart contracts and accounts. This is particularly useful for developers familiar with Ethereum tooling and ecosystem.

Mirror Node Explorer: A web-based interface that allows developers to audit transactions, accounts, and other network activities visually.

Setup and Configuration

Single-node configuration simulates the network's functions on a smaller scale (on a single node), ideal for debugging, testing, and prototype development. Multi-node configuration distributes multiple instances of the Hedera network nodes across a single machine using Docker containers, intended for advanced testing and network emulation.

☐ Single Node Configuration

☐ Multinode Configuration

Operational Modes

Local Node offers two modes depending on a developer's needs:

<details>

<summary>Full mode</summary>

Full mode is activated with the --full flag, and the system is designed to capture and store comprehensive data. Here's how it works:

Data Upload: Each node within the network generates record stream files during operation. Record stream files are a sequence of transaction records grouped together over a specific interval. The Hedera network periodically consolidates these transaction records into stream files, which are then made available to the network nodes and mirror nodes. In full mode, these files are systematically uploaded to their own directory within the minio bucket. MinIo is an object storage platform that provides dedicated tools for storing, retrieving, and searching blobs. This process is managed by specific uploader containers assigned to each node, namely:

- record-streams-uploader-N (contains record streams)

- account-balances-uploader-N (contains account balances files)

- record-sidecar-uploader-N (contains a list of TransactionSidecarRecords that were all created over a specific interval and related to the same RecordStreamFile.

</details>

<details>

<summary>Turbo mode</summary>

Turbo mode is the default setting when running the local node. This mode prioritizes efficiency and speed, with the following key characteristics:

Local Data Access: Instead of uploading data to the cloud, record stream files are read directly from their corresponding local directories on each node. This method significantly reduces latency and resource consumption, making it ideal for scenarios where immediate data access and high performance are prioritized

over long-term storage and external accessibility.

</details>

With these two options, users can tailor the local node's operation to suit their needs best, whether ensuring comprehensive data capture and backup or optimizing for performance and speed.

Solo

Solo offers an advanced private network testing solution and adopts a Kubernetes-first strategy to create a network that fully mimics a production environment. Explore the Solo repository [here](#).

More info coming soon...

Additional Resources

- Hedera Local Node Repo
- Docker Compose Documentation
- Run a Local Node in Gitpod \[tutorial]
- Run a Local Node in Codespaces \[tutorial]

single-node-configuration.md:

Single Node Configuration

Using Single Node Configuration

Single node configuration simulates the network's functions on a smaller scale (single node), ideal for debugging, testing, and prototype development.

<details>

<summary>Single Node Mode Requirements</summary>

Ensure the VirtioFS file sharing implementation is enabled in the docker settings.

.png)

Ensure the following configurations are set at minimum in Docker Settings -> Resources and are available for use:

CPUs: 6

Memory: 8GB

Swap: 1 GB

Disk Image Size: 64 GB

Ensure the Allow the default Docker sockets to be used (requires password) is enabled in Docker Settings -> Advanced.

Note: The image may look different if you are on a different version

</details>

Starting and Stopping Node

Before launching the network commands, confirm that Docker is installed and open on your machine. To stop your local node, use the following npm or docker commands. Before proceeding with this operation, make sure to back up any manually created files in the working directory.

<details>

<summary>npm commands</summary>

{% code overflow="wrap" %}

bash

npm command to start the local network and generate accounts in detached mode

npm run start -- -d

npm command to stop

npm run stop

npm command to restart node

npm run restart

{% endcode %}

</details>

<details>

<summary>docker commands</summary>

bash

Docker command to start the local network. Does not generate accounts

docker compose up -d

Docker command to stop services

docker compose stop

Docker command to restart local network

docker compose restart

Docker command to stop local network and remove containers

docker compose down

</details>

Alternatively, run `docker compose down -v; git clean -xfd; git reset --hard` to stop the local node and reset it to its original state. The full list of available commands can be found [here](#).

Single Node Mode Diagram

The following diagram illustrates the architecture and flow of data in single node mode.

<figure><figcaption><p>Single node mode diagram</p></figcaption></figure>

mainnet-access.md:

cover: ../../.gitbook/assets/HH-Eco-Cat-Hero-Desktop-R1 (1).webp

coverY: -625.8620689655172

Mainnet Accounts

To interact with and access the various Hedera Mainnet services such as accounts, topics, tokens, files, and smart contracts, you will need a Hedera account. Your Hedera account also holds a balance of HBAR, which can be used to make transaction fee payments or transfers to other accounts.

Create free mainnet accounts by visiting any of these wallet providers:

https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable
https://atomicwallet.io/		Private Key Viewable		Private Key Viewable

```
href="https://wallawallet.com/">https://wallawallet.com/</a></td></tr></tbody></table>
```

Feature	Description
-----	-----
<input checked="" type="checkbox"/> Private Key Viewable	You have access to the private key associated with the mainnet account the wallet created for you
<input checked="" type="checkbox"/> SDK-compatible Passphrase	The passphrase created by the wallet is compatible with the SDKs and can be used to recover the private keys for the account the wallet created for you

Create new mainnet accounts

Once you have obtained your mainnet account from a supported wallet, you can use the SDKs to create additional mainnet accounts.

To do this, you will need to point your Hedera client to mainnet (`Client.forMainnet()`) and use the `AccountCreateTransaction` API to create a new account. The transaction fee payer (referred to as the operator in the SDKs) information should be set to the mainnet account you created from one of the above wallets (`setOperator(<mainnetAccountId, mainnetAccountPrivateKey>)`).

```
{% content-ref url="../../../sdks-and-apis/sdks/accounts-and-hbar/create-an-account.md" %}
create-an-account.md
{% endcontent-ref %}
```

README.md:

```
---
description: Join Hedera Mainnet
---
```

Mainnet

Overview

The Hedera mainnet (short for main network) is where applications are run in production, with transaction fees paid in HBAR. Transactions are submitted to the Hedera mainnet by any application or retail user; they're automatically consensus timestamped and fairly ordered.

Data associated with Hedera's services and stored on-chain can be queried by any Hedera account. Every transaction requires payment in the form of a transaction fee denominated in tinybars ($100,000,000 \text{ th} = 1 \text{ h}$). You can learn more about transaction fees and estimate your application costs [here](#);

If you're looking to test your application (or just experiment), please visit [Testnet Access](#). The Hedera testnet enables developers to prototype and test applications in a simulated mainnet environment that uses test HBAR for paying transaction fees.

```
{% hint style="warning" %}
Transaction Throttles\
Transactions on the Hedera Mainnet are currently throttled. You will receive a
"BUSY" response if the number of transactions submitted to the network exceeds
the threshold value.
{% endhint %}
```

Network Throttles

Network Request Types	Throttle (tps)
-----	-----
Cryptocurrency Transactions	<p>AccountCreateTransaction: 2 tps</p><p>AccountBalanceQuery: unlimited</p><p>TransferTransaction (inc. tokens): 10,000 tps Other: 10,000 tps</p>
Consensus Transactions	<p>TopicCreateTransaction: 5 tps</p><p>Other: 10,000 tps</p>
Token Transactions	<p>TokenMint:</p>125 TPS for fungible mint50 TPS for NFT mint<p>TokenAssociateTransaction: 100 tps TransferTransaction (inc. tokens): 10,000 tps</p><p>Other: 3,000 tps</p>
Schedule Transactions	<p>ScheduleSignTransaction: 100 tps ScheduleCreateTransaction: 100 tps</p>
File Transactions	10 tps
Smart Contract Transactions	<p>ContractExecuteTransaction: 350 tps ContractCreateTransaction: 350 tps</p>
Queries	<p>ContractGetInfo: 700 tps ContractGetBytecode: 700 tps ContractCallLocal: 700 tps FileGetInfo: 700 tps FileGetContents: 700 tps Other: 10,000 tps</p>
Receipts	unlimited (no throttle)

README.md:

```
---
description: Hedera network fees
---
```

Fees

The Hedera testnet fees tables found below offer a low-end estimate of transaction and query fees for all network services. The tables below contain USD, HBAR, and Tinybar (t̄h) values per each API call. All operation fees on the Hedera testnet are paid in test HBAR, which is freely available and only useful for development purposes.

Fee estimates are based on assumptions about the details of a specific API call. For instance, the fee for an HBAR cryptocurrency transfer (CryptoTransfer) assumes a single signature on the transaction and the fee for storing a file assumes a 48-byte sized file stored for 30 days. Transactions exceeding these base assumptions will be more expensive; we recommend increasing your maximum allowable fee to accommodate additional complexity.

Mainnet Fees

Mainnet transaction and query fees can be estimated using the Hedera Fee Estimator. The Fee Estimator allows you to determine fees (in both USD and HBAR, using the current exchange rate live on the mainnet) for individual transactions & queries based on their characteristics, as well as projected costs based on expected volume for those transactions. The estimations may not be 100% accurate and the underlying prices are subject to change without prior notice.

HBAR Denominations and Abbreviations

Denominations	Abbreviations	Amount of HBAR Cryptocurrency
-----	-----	-----
gigabar	1 Għ	= 1,000,000,000 ħ
megabar	1 Mħ	= 1,000,000 ħ
kilobar	1 Kħ	= 1,000 ħ
hbar	1 ħ	= 1 ħ
millibar	1,000 mħ	= 1 ħ
microbar	1,000,000 μħ	= 1 ħ
tinybar	100,000,000 tħ	= 1 ħ

Transaction and Query Fees

All fees are subject to change. The fees below reflect a base price for the transaction or query. Transaction characteristics may increase the price from the base price shown below. Transaction characteristics include having more than one signature, a memo field, etc. Please reference the Hedera fee estimator to estimate the transaction or query fee.

Cryptocurrency Service

Operations	USD
CryptoCreate	\$0.05
CryptoAccountAutoRenew	\$0.00022
CryptoDeleteAllowance	\$0.05
CryptoApproveAllowance	\$0.05
CryptoUpdate	\$0.00022
CryptoTransfer	\$0.0001
CryptoTransfer (custom fees)	\$0.002
CryptoDelete	\$0.005
CryptoGetAccountRecords	\$0.0001
CryptoGetAccountBalance	\$0.00
CryptoGetInfo	\$0.0001
CryptoGetStakers	\$0.0001

Consensus Service

Operations	USD
ConsensusCreateTopic	\$0.01
ConsensusUpdateTopic	\$0.00022
ConsensusDeleteTopic	\$0.005
ConsensusSubmitMessage	\$0.0001
ConsensusGetTopicInfo	\$0.0001

Token Service

Operations	USD
TokenCreate	\$1.00
TokenCreate (custom fees)	\$2.00
TokenUpdate	\$0.001
TokenFeeScheduleUpdate	\$0.001
TokenDelete	\$0.001
TokenAssociate	\$0.05
TokenDissociate	\$0.05
TokenMint (fungible)	\$0.001
TokenMint (non-fungible)	\$0.02
TokenMint (bulk mint 10k NFTs)	\$200
TokenBurn	\$0.001
TokenGrantKyc	\$0.001
TokenRevokeKyc	\$0.001
TokenUnfreeze	\$0.001
TokenPause	\$0.001
TokenUnpause	\$0.001
TokenWipe	\$0.001
TokenReject	\$0.001
TokenGetInfo	\$0.0001
TokenGetNftInfo	\$0.0001
TokenGetNftInfos	

	\$0.0001
TokenGetAccountNftInfos	\$0.0001
TokenUpdateNfts (updates metadata of 1 NFT)	\$0.001
TokenUpdateNfts (update multiple NFTs in a single call)	\$0.001

Schedule Transaction

Operations	USD (\$)
ScheduleCreate	\$0.01
ScheduleSign	\$0.001
ScheduleDelete	\$0.001
ScheduleGetInfo	\$0.0001

File Service

Operations	USD (\$)
FileCreate	\$0.05
FileUpdate	\$0.05
FileDelete	\$0.007
FileAppend	\$0.05
FileGetContents	\$0.0001
FileGetInfo	\$0.0001

Smart Contract Service

Operations	USD (\$)
ContractCreate	\$1.0
ContractUpdate	\$0.026
ContractDelete	\$0.007
ContractCall	\$0.05
ContractCallLocal	\$0.001
ContractGetByteCode	\$0.05
GetBySolidityID	\$0.0001
ContractGetInfo	\$0.0001
ContractGetRecords	\$0.0001
ContractAutoRenew	\$0.026

Miscellaneous

Operations	USD (\$)
EthereumTransaction	\$0.0001
PrngTransaction	\$0.001
GetVersion	\$0.001
GetByKey	\$0.0001
TransactionGetReceipt	\$0.0000
TransactionGetRecord	\$0.0001

transaction-records.md:

Transaction Records

Understanding Transaction Records – Remittances & Fees

Once a transaction has been successfully processed by the Hedera network into a consensus state or not, the network nodes create either a "record" or a "receipt," respectively, for that transaction, indicating its status and impact.

A key component of the information within a record for a transaction is how the transaction changed the balances of those Hedera accounts that were involved. An account's balance can change due to a transaction either because of a fee (paid or received) or some other intentional transfer – which we refer to here as a 'remittance'.

Account Balances

Every transaction submitted to Hedera will cause the balances of a set of accounts to change – either because

1. The transaction directly directed specific balances to be changed, e.g. Alice sent 1000 hbars to Bob with a CryptoTransfer.
2. The transaction indirectly caused balances to change, e.g. execution of a ContractCall caused HBARS to be sent from the Smart Contract's account to others.
3. Fees are paid for the processing of the transaction into a consensus state and the persistence of that changed state.
4. A fee is paid for the persistence of a 'record' for that transaction for a longer period of time than the default.

A transaction can cause a given account's balance to change for any of the above reasons or, more generally, for a combination of the above reasons.

There are, in general, 5 types of accounts associated with a transaction:

1. Senders – the accounts from which HBARS are being sent
2. Receivers – the accounts to which HBARS are being sent
3. Payer – the account that pays the fees associated with the transaction.
4. Network – the Hedera account that receives the component of the fees that compensate the network for processing the transaction.
5. Node – the account of whichever node submits the transaction to the network for consensus

Notes

For any transaction, the sum of transfers out of all accounts will always be equal to the sum of all transfers into all accounts.

The payer, in general, is different than either the sender or receiver.

Nevertheless, a typical case is that the sender will also be the payer.

Not all transactions have a sender or receiver as there is no remittance aspect to the transaction, e.g. a FileCreate or ConsensusSubmitMessage transaction will have a fee but no associated remittance.

A single CryptoTransfer can have multiple senders and multiple receivers.

A remittance can be the number of HBARS a CryptoTransfer directs be moved, the amount of HBARS a CryptoCreate directs be funded into the new account, or the amount of HBARS in an account to be deleted, with those funds moved into another account.

A remittance will need to be authorized by the owner of those HBARS.

An account owner can specify thresholds for transfers in and out of that account. If a transaction causes an account's threshold to be triggered, then the record for that transaction will persist for 25 hours and not the default 3 minutes.

The account owner that specified the threshold will pay a threshold fee – distinct from the fee for the transaction itself – for that extra storage time.

It is account 0.0.98 that receives the component of the transaction fee that compensates all the nodes for their work in processing the transaction into consensus

0.0.98 also collects any threshold record fees

As of early February 2024, there are 31 nodes with account numbers in the range of 0.0.3-0.0.4698971.

While accounts 0.0.98 and the node accounts are special with respect to receiving fees, they can also send & receive HBARS and, as such, could be the sender or receiver of a transaction.

Scenarios

We explore scenarios below and how the HBARS flow between accounts for each.

Case 1 - Generic

In the most generic case, a sender is making a remittance to a receiver, and a separate account pays the associated fee. The size of the fee will depend on the nature of the transaction – uploading a large file will cost more than a simple

crypto transfer.

The amount of that fee is split between account 0.0.98 (a special Hedera account that represents the network) and the specific node that submitted the transaction.

Case 2 - Fees only

Many transactions do not allow for an explicit remittance, for instance, a FileCreate or a ConsensusSubmitMessage. For such transactions, the only changes to account balances will be due to the fee for that transaction.

As before, the fee for the transaction is split between 0.0.98 and a node.

Case 3 - Sender account pays fees

It will often be the case that the fee for a CryptoTransfer sending remittance from a sender to a receiver is paid for by the sender. In this case, the balance for the sender will decrease by the sum of the remittance and the fee.

In principle, the receiver could pay the fee as well.

Case 4 - Sender account has a threshold that is crossed

Account owners can specify thresholds for their accounts so that any transfer in/out of the account that exceeds these thresholds will cause the created record to be persisted for 25 hours and not the default 3 minutes. The account that stipulated the threshold will pay a threshold fee for this prolonged storage of the record.

In this example, the threshold is specified on the sending account, and so it will be that account that pays this threshold fee. That account's balance consequently decreases by the sum of the remittance and the threshold fee.

Account 98 receives the sum of the network, service, and also this threshold fee.

Not shown here but if it were also the case that the sender was paying the transaction fee (as above) then the balance of the sender's account would decrease by the sum of the remittance, the transaction fee, and this threshold fee.

Case 5 - Receiver account has a threshold that is crossed

The receiving account can also have a threshold set that, if surpassed by the amount of HBARS being transferred into the account, will cause the record to be stored for 25 hours.

Threshold records can be particularly useful to receivers as a receiver may not be aware of the remittances sent to them (as they are not necessarily involved in the signing of the transaction, as is the case for the sender). The longer-lived threshold records allow an account owner to query on a daily basis for the records for any and all remittances they've received over the past 24 hours that they might not otherwise be aware of.

The receiving account will pay the associated threshold fee for this longer

storage period of the record.

The net balance change of the receiving account will therefore be the remittance minus the threshold fee.

Account 98 receives the sum of the network, service, and the threshold fees.

The transaction fees are not impacted by the threshold fee being paid.

If the value of the remittance is less than the threshold fee, the transaction will fail.

Case 6 - Node account is receiver

Nodes may receive remittances like any other Hedera account.

As a specific example, clients compensate nodes for responding to a query by including within the query a CryptoTransfer that, when submitted to the network, will compensate that particular node with a suitable remittance.

In this scenario, the node account's balance will increase by the sum of the node fee it receives for processing the CryptoTransfer plus the value of the actual remittance that pays the node for the query response.

Transaction Records

After Hedera Mainnet nodes process a transaction into consensus state, the details are published to the outside world in a 'transaction record' that the nodes create and make available. Clients retrieve records and analyze the data within to verify the consequences of transactions, for instance the consensus timestamp that was assigned and how the associated account balances changed as a result of the transaction.

When retrieved from a mirror node and not the mainnet, the transaction that resulted in a given record will also be available. It is therefore this combined data structure that provides the richest set of information for analysing and differentiating between remittances and fees.

The flow of information is shown below:

A client that retrieves the pair of a transaction and its associated record may want to distinguish between remittances and fee components for the transaction - that is, what part of an account's balance change was due to transaction fees, what part due to a threshold fee, and what part due to a remittance.

There is sufficient information in the combination of transaction and corresponding record to allow a client to unambiguously make such a distinction for each account.

A transaction record has a transfer list data structure that describes how HBARS moved between accounts as a result of the transaction.

In the R3 (the release prior to the update of February 10, 2020) version of the node software, there might be multiple transfers for each account involved in the transaction. For instance, there could be separate transfers indicating the 0.0.98 account receiving fees, which added up to the correct total fee.

Additionally, in R3

1. The determination of whether a threshold was exceeded for each account was made for each transfer. Consequently, a single account paying both a remittance and fees could pay for multiple threshold records if the threshold was set very low.
2. The order of the transfers in the R3 format was not predictable.

We have changed the record transfer list format in the R4 (the release of February 10, 2020) node software to address the above issues and to be more consistent and concise.

In the R4 release, the record transfer list shows, for all transaction types, only a single net transfer in or out for each relevant account.

The comparison of transfers in/out to an account's thresholds is now made on that net transfer, and not on the constituent transfers that summed to the net. Consequently, any account will pay once for only a single threshold fee and not multiple times. This is cheaper for the user.

The change between R3 and R4 is shown below, for a representative transaction in which account 0.0.1002 is sending a remittance of 10,000 tinybars to account 0.0.1001 and both sender and receiver have thresholds of 1,000 tinybars set on their accounts.

As the remittance value exceeds these thresholds, both sender and receiver will pay a threshold fee.

R3

```

```

R4

```

```

In the R4 format, the record transfer list no longer has multiple transfers for the different accounts – each account has only a single transfer with a value that reflects the sum of the various transfers that impacted each account.

While the R4 format is more concise than the R3 format, some clients may want to determine the component transfers - that is to break out remittances, node fees, threshold fees, and other transaction fees. To facilitate this analysis, Hedera plans to add support to the mirror node REST API to allow a client to request either the default aggregated transfer list, or instead an itemized list of transfers (similar to the R3 format).

README.md:

Mainnet Consensus Nodes

Hedera networks are comprised of two types of nodes: Consensus and Mirror nodes. The Hedera Mainnet consensus nodes are currently permissioned; operated by the Hedera Governing Council. Consensus nodes receive transactions from clients, charge transaction fees, and contribute to consensus being achieved. Mirror nodes are permissionless, and store the history of transactions for optimized queries. Mirror nodes do not submit transactions to the network from clients nor do they participate in consensus.

```
{% content-ref url="../../../../../core-concepts/mirror-nodes/" %}  
mirror-nodes  
{% endcontent-ref %}
```

Mainnet Node Address Book

The address book contains the list of consensus nodes that can submit transactions to mainnet for a user. For each node, the node account ID is the ID of the node (0.0.x) and the node address is the IP address and port for that node. The mainnet address book file ID on mainnet is 0.0.102. The certificate hash value is the SHA384 checksum of the TLS certificate presented from the various IP addresses on port 50212 in PEM format.

For node status, please visit the Hedera status page [here](#).

{% hint style="info" %}

Note: The TLS port (50212) is currently not supported by Hedera SDKs.

{% endhint %}

Node		Node Account ID		IP Address		Port		Node Certificate Thumbprint								
LG	0	0.0.3	35.237.200.180	34.239.82.6	13.124.142.126	15.164.44.66	15.165.118.251	50211, 50212	(TLS)							
Swirls	1	0.0.4	35.186.191.247	3.130.52.236	50211, 50212	01d173753810c0aae794ba72d5443c292e9ff962b01046220dd99f5816422696e0569c977e2f169e1e5688afc8f4aa16	Worldpay	2	0.0.5	35.192.2.25	3.18.18.254	23.111.186.250	74.50.117.35	107.155.64.98	50211, 50212	(TLS)
Wipro	3	0.0.6	35.199.161.108	13.52.108.243	13.235.15.32	104.211.205.124	13.71.90.154	50211, 50212	(TLS)							
Nomura	4	0.0.7	35.203.82.240	3.114.54.4	50211, 50212	b8707dd891621b10fce02bd6ea28773456f008b06b9da985ae2da1ad66be8237cf831fc5b8b4fea54595179e9735d5d2	Google	5	0.0.8	35.236.5.219	35.183.66.150	50211, 50212	(TLS)			
Zain	6	0.0.9	35.197.192.225	35.181.158.250	31.214.8.131	50211, 50212	(TLS)									
Magalu	7	0.0.10	35.242.233.154	3.248.27.48	179.190.33.184	50211, 50212	(TLS)									
Boeing	8	0.0.11	35.240.118.96	13.53.119.185	7031f1541dbd7b6fe7da70240265820d37f7c529a93348f50d78421b18797e7b15											

cde7d0e5f057c884b87d093e7d38f5</td></tr><tr><td>DLA Piper</td><td align="center">9</td><td align="center">0.0.12</td><td><p>35.204.86.32</p><p>35.177.162.180</p></td><td>50211, 50212 (TLS)</td><td>200cdb854f985aa6d6bd159a24a034eabe90d838a8480f8fb0e6e92eb5c57be2ecbe54a32c71ae4f971e3c36f2f70c9</td></tr><tr><td>Tata Communications</td><td align="center">10</td><td align="center">0.0.13</td><td><p>35.234.132.107</p><p>34.215.192.104</p></td><td>50211, 50212 (TLS)</td><td>07b77ce284f7ebb5beb07b105375af55d228a765e1a84587eb1d1f1b0675c38a1d1512370e56f21c8ed5eadefb3779a9</td></tr><tr><td>IBM</td><td align="center">11</td><td align="center">0.0.14</td><td><p>35.236.2.27</p><p>52.8.21.141</p></td><td>50211, 50212 (TLS)</td><td>50afa448a3a78b615f49fef577bd7b62b13082eb552cf8895109e0e5438f79ac8aed2f2a48e2f570490746f4c439104b</td></tr><tr><td>Deutsche Telekom</td><td align="center">12</td><td align="center">0.0.15</td><td><p>35.228.11.53</p><p>3.121.238.26</p></td><td>50211, 50212 (TLS)</td><td>c6713d87e3c1f6859a3a3663ebb1b7e1bd1da14fcf076ce51d36464b5682a5df7cfadd440197564f498842313091c2bd</td></tr><tr><td>UCL</td><td align="center">13</td><td align="center">0.0.16</td><td><p>34.91.181.183</p><p>18.157.223.230</p></td><td>50211, 50212 (TLS)</td><td>a53c8eb70bdd89edb6be5fec50cfabc081002a477af478eefa355ea6ee572e4ae50e84988f8ea2c068afa78967b2caf0</td></tr><tr><td>Avery Dennison</td><td align="center">14</td><td align="center">0.0.17</td><td><p>34.86.212.247</p><p>18.232.251.19</p><p>34.86.212.247</p><p>18.232.251.19</p></td><td>50211, 50212 (TLS)</td><td>b5b28e6f57240730cda802c56e53b8837a7d63dab9b3b3ab156a04d9e2e6e24b6790c38a5fd335be67a951b77beec748</td></tr><tr><td>Dentons</td><td align="center">15</td><td align="center">0.0.18</td><td><p>141.94.175.187</p><p>34.87.86.242</p><p>18.136.143.176</p></td><td>50211, 50212 (TLS)</td><td>eb976995e5d2a9da69c44b06df3c29ecca5eba5008f054077c0ee87b08813314c4fd91ec834d3b868fb7ee7794f4cbeb</td></tr><tr><td>Standard Bank</td><td align="center">16</td><td align="center">0.0.19</td><td><p>34.89.87.138</p><p>18.168.4.59</p><p>13.246.51.42
13.244.166.210</p></td><td>50211, 50212 (TLS)</td><td>d5ce6f4378d6d547239486efd6a702a74fca9e965723f0ea43eefa32be98179508c42185fa23750d20e8c0f1ca1d563a</td></tr><tr><td>Australian Payments Plus</td><td align="center">17</td><td align="center">0.0.20</td><td>34.82.78.255</td><td>50211, 50212 (TLS)</td><td>b0d760b75396bd80292cf751dc43832a01426fc79e2940aa9d2839525ffda1db04bd47d4e2ed3b46088373800504ce13</td></tr><tr><td>EDF</td><td align="center">18</td><td align="center">0.0.21</td><td>34.76.140.109
13.36.123.209</td><td>50211, 50212 (TLS)</td><td>c32b80403febe31c4ba1eb46cfa93cbc9169f28597a30ca01365cb3e179672a755a38450ce6aaabf40ad9e36048fd8d4</td></tr><tr><td>Shinhan Bank</td><td align="center">19</td><td align="center">0.0.22</td><td>34.64.141.166
52.78.202.34</td><td>50211, 50212 (TLS)</td><td>73d89b53f7cc1ec674af28ad521faf08d7a018166469efc845154ac41108a9d8129a8830031a19a5751a2d79cbc47520</td></tr><tr><td>Chainlink Labs</td><td align="center">20</td><td align="center">0.0.23</td><td><p>35.232.244.145
3.18.91.176</p><p>69.167.169.208</p></td><td>50211, 50212 (TLS)</td><td>903388fe34e8bd8cae28eec4c6e8fdd2035b1dcb5c9f45b32b4b17d658c688b81d2330689564c371d477f68b4d2b3959</td></tr><tr><td>LSE</td><td align="center">21</td><td align="center">0.0.24</td><td>34.89.103.38
18.135.7.211</td><td>50211, 50212 (TLS)</td><td>416f95bfee63fdd45bfc1007cbec3d544fac6b3039fe1d00317aff4c9c9e67918b

28c48bddb23c4ca11131cd8260b3ed	IIT Madras
align="center">22	
align="center">0.0.25	<p>34.93.112.7</p><p>13.232.240.207</p>
50211, 50212	
(TLS)	8852cb8dcddab369e5719cdbac5e5f50098f882176412ad711e2b03ac6e9fe7035826fd47c12a12742d2c00d5075753f
align="center">23	DBS
align="center">0.0.26	<p>34.87.150.174</p><p>13.228.103.14</p>
50211, 50212	
(TLS)	8852cb8dcddab369e5719cdbac5e5f50098f882176412ad711e2b03ac6e9fe7035826fd47c12a12742d2c00d5075753f
align="center">24	ServiceNow
align="center">0.0.27	<p>34.125.200.96</p><p>13.56.4.96</p>
50211, 50212	
(TLS)	9e51d01fd202f55080e5ddc4c5a9822747c66a857b0d4070fc20b3f508c5a20239877a219b761825349af5f30010a83d
align="center">25	Ubisoft
align="center">0.0.28	<p>35.198.220.75</p><p>18.139.47.5</p>
50211, 50212	
(TLS)	b02c9e85f49da18147fd1353773baf0e71d72e22a007f1df9051aa8a4761ae87ad4f90ea4c3409811a1c4777cc2fc61f
align="center">26	abrdn
align="center">0.0.29	<p>34.142.71.129</p><p>54.74.60.120</p><p>80.85.70.197</p>
50211, 50212	
(TLS)	fff800ea4280d62c9c1ff333cf430194e0f8af282b813b45211328533cf72f9f14644c0604aacf12b165b5a6b059acc3
align="center">27	Dell
align="center">0.0.30	<p>35.234.249.150</p><p>34.201.177.212</p>
50211, 50212	
(TLS)	9773691c2995551932e052e7dc80532b31d8470ec2db37956ce88036817965436300f00d95ba5a750c2b7891c8b66a9a
align="center">28	COFRA Holding
align="center">0.0.31	<p>217.76.57.165</p><p>34.107.78.179</p><p>3.77.94.254</p>
50211, 50212	
(TLS)	f7313bc08b9d4bc794cb1404d1f482a73ac5fe0c0293b5c73647de3841b0591066624e9922172d54212b1b9e6a790908
align="center">29	Hitachi
align="center">0.0.32	<p>34.86.186.151</p><p>3.20.81.230</p>
50211, 50212	
(TLS)	566be8269cf6b1399dca82088d60755d0edc05a1b1f48f3fb6dbb30c8425f3514ba5ca0be3c546d11990586618f3f665
align="center">30	Mondelēz International
align="center">0.0.33	5.199.164.101
50211, 50212	
(TLS)	3e3175ea8aa2ec755bd346275317efbf3630ffff31c6eb08c7c911d4d24834518fd8bcd474cf9cf5225b4c266d4f0e0e
align="center">31	BitGo
align="center">0.0.34	64.185.230.146
50211, 50212	
(TLS)	eadd72fcf60fab34228c729a6d2584ad6542c2e4e785a351d31ec83250d482d40dea6177fcdcbcd6e01acb3e80c9b0ca8

Mainnet Node Public Keys

Below you will find the mainnet node public keys found in the mainnet address book file 0.0.102. You can also access the address book by using the state proof alpha API or SDKs as well. The public keys stored in the address book are hex-encoded keys (x509).

Node Account ID	Public Key Modulus
0.0.3	09098865def2f2ab376c7f0f738c1d87a27ac01afd008620c35cb6ebfcbb0c33003193a388c346d3023172701

2193bb76fd3004b864312c689ef5213cbb901101509deab94f26a732e637929da4c4cb32517e3adb
b3811d50ac4c77c1fce8b651606215f34707f3e7265545e58c894609e28376bdb7775fe30439e0e1
592fdcb0c3ee1c305773d072a6b8957eafce1a11be965edaff3843366cb6a44ec25a890106e62475
67f76b550fda482baec6307d698ec88841fd66f23f210e47b8a9dcba6ba4e1fa716db33c80e30819
496dcb5e5609fb6e7c615379bdded427e9231b9254c2baf943608a86d698ae9a3c8639df887d6f6b
5a71385d24338d911a212bf71f1e2acc8b186b96ec8e69c86b6d058217776a09c9c68953edb59165
78b5a263b2f469e3b0c07ead71a447eea7f8fc1bb8074255567b7f0bd1e6afb0358718c98b429e2
4b2298596fc76cf6af396ca9434d7926ec7d37d4b92af56d45feff8196095224a916c1ffe6b667e2
55fc3ac8cccef920dc044b25003132b87806742f</td></tr><tr><td>0.0.4</
strong></td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009131aa3
68f9345229f97b6259cccaffea23e00cd5ead02e3f696c1e714ee3939dad860e38bf95a2974f9eb4
8e9343f8aac405ea955d05323e117b3b1c94813a3af42fe8082c3d43baf1bd4d8367e93db00ad696
e627a1036ae534f011ead5e56f37a6ffe44b6b9e099401192ad560a0346b41a810095f5f2d7fd32d
6eeb655ba758c6b526c129386af7197c7a53ae603d622832254961f16d0efa8079a768561888be73
3492217956bbcafaeb6135c5fbb2484d5b4a5fdf0336ac02e26c1652c1bd8eaf30dae1d6d3eb00f
7b4fab8d6478fe8d95eb911df966a0dea4e522db76b8966570ecc5af09516424f0af5f8ee66e386d
5650713997169ac37573bf52fd058de95ab2ff68e68111ab23405ea964b2bb88d02c0f1caed71ecd
d4e4e408594876fdb8500bc55c7ba02066e05ab98d9f7e0466d9702eb57ee3722f8fcc85a75505ff
3262170288b788723adb97e4de5620cc90ead1382fcd7571889fefb11e6771bc3f6f3feb19c7ac54
2878d03a90270526c3eed2494eff54e153ca9f6890203010001</td></tr><tr><td>0.0.5</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100b2ccac6
5ad0fc7645a817bfabc487ad7e41311e7a3198b37fb842d84c395b3f67d6bd848f10c6f03c290e8f
7daa8d001a8441dc352a19160a3193e68b82edf19ae67693a9a33d4cb87e789a1070715515ea772c
aa8b86a569b91c5450835d9c354f0dacec97fe77091b45b147698b7f8601422dcd2261e928de4dac
9c42dcbafdf96c07233ba3027076f37c969e8ed30b6b5d8f5034be7d92c596f8be861e51fcc3a242
bf9d8be9e2a9e8e0f155ebcff23effa7cd57c10542811d80776c9585526fdb0eaa34ee1955d51119
390fe873e4c04dedd29165884b98b46308788ae7fc4d4aa4a8fc9bc2674ba321493b624455ad410c
1de71bc95d1d91fa0f201418a795e309eaf297b699bf27c9fa2763cd59ceb021e16b8200c1060f28
17fd83cfc767183489461e3599291b380d6e939baa4b19232a6a272dde651f8046fdc34db276a777
d6fb2bec3255b2cc244b4af566b105f30c6506ddae0eb3deddcf947bcb9c60e000984f3b4a8c6c4e
d4bf90bc1932b7f94dc3ae6b360008eb902040f9b0203010001</td></tr><tr><td>0.0.6</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a3e37b7
6c6cd5f6622d6924444d12c677c395f2b5902f3bb98b8a8b5055a707706ca028cd75060a2d8702d2
d8b04947bdcfe0a8c141aa2844b1e06e66190012e8b6326ab0fa317973bc7cb4d2949f2108aa04c4
b0c91baa5728f5b5622ec75abf578a1f7b41ede2a67ebd69c18e581fdf9c6020ac0de9ca2c31f0c6
469003311fbb5ce7db49c787e1a7d27aa425ee7b84da7e66939f9c80d0e82fce55e02dfc8b5c7841
8a26aa43650698719bafcecf0bd49000addcfa405708bdbbfbb19749d22dab007e44d45ea23b106f
8834c152e9062d4cf24ff25356c7eb3729105393fb49bab904a02f0f0bb417cd919d352890128e6
bbff4fac9f50de118a974f2a6dd01e032a79b178f60fa1fcbdd02b5704fb46295c15190816373edd
6635c856978f1b9503f1f73b4b0be8aba2ed1feead59953bf82efde93a3471abd55cda3ba8a673fb
b3799749fb006d003f0e63f665c3461d2a7b29dc8b204ba59a65668a46ae2878f00d1f9490df9e28
0febf4315ea04eaa568a3a9fd48c62c63b6ecda690203010001</td></tr><tr><td>0.0.7</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c4561e3
c278cd650e80c413ca44423c1c3c13cf1475f6f6976d597ae432b49ab42086b79b841326054b8b3d
cf57d8fcd79bfc058183ca24cd4c1cbc574ed1117e2f5b7b3c63ce7b06d9b4efcf7375637b41fe6f
53c811b9de6143f3a52957cdf956775120b33703ff57621407ab9575bc2d35c0d44f0983fc1ef63a
4ff5209f070c92af106295601c96bcd064ec190197019c6811c4c8dd80cb4f4ac71f9ad76e7ac89
456fbf4f011f90abd2d90536e8234651f6bef927e3d5d8b7bf459050983beca3abef2a9d97af3457
72a7740e9699275b018ea0df286add6ce923ef908fbe762a75f21116862db44d3dca1d44b4d2e8dc
1066c5006bb5a7d954ad255d4b603273475e511aeb485d069a067c0ab5c24538c933c06b5a6aef9
4005c2915213e4ccdae6c942f6272f9dd5282d6b890f1f20efd2399cd674924fa57046ac6da32e73
951a73113e91fc2b7ff29e4851b83ff39f83ba9ec6f08cefdbb6cbbbffabfdfaa91d930f7200da48
137c394cbd13e701ecdc2616fd21bad681aa4f0010203010001</td></tr><tr><td>0.0.8</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a1c4077
154303cc72c4fb7692c3f94251bdec1239a1f7a8972abe91a35323fbeca625a7ffae6406c855dc2a
f2110900b0df0e6e6db76364dfa1ffe85eda567936e2985b85634a32aa52a6599dd6c30be1f7a6c5
b8f5eeca2621d8a459682fcd2dbaad1561d11f33fccb7f5500ac568d165dbeaace3286d2894f641
29d781d6c72fd7d599c9e1d3af4aa433c23b910fae4c4841641f61526ad787e8ea539874167e9d3a
73cc0fb156429d15ec763a6d0f06115a79b9af783d77b98d83096aa4743f97408d9e14bcf4ddffe4

591768847b40cb8da7ca375256d2b935d095fe252fae81ff6e37f84d7a90d7e570a4f8ef3c7d766e
eda472f0920199015a8908259a873c5454fcbdbdcad2e528de85455b4083c7dc4adc5a988e0cddfdc
159d5d712abd544aa73ec029089814c98a44f26fc0644659c183e3184aa272f8d1dc0bfa3e0a5604
84cb055ba4dbb5cc339ec80bd11d642dc3a702e8c703ab2193084d9bd63f0dfe12a433c2576eaf78
1cfad867ef70bda61768b2bef14f50c6c3b8b096f0203010001</td></tr><tr><td>0.0.9</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810093a215c
c4a7a722cae9c13abd636df99ccec6af9db46b69fa516716ef50ce2490a981e09ab019ca2cb4681
1b5b619d1bd1d5ee6f46a42c777cbdee642a1484ecdf5ddd3729642c38c6d43a8858874475f58244
43664c04dfed9b89045fb085e25c3efcb4841733eff7c529c139e69350c2cd79b2c8d19679a712e4
e8cafd3267541b832b3e10a01255def69df1e9d3b8d8eaf0311de67d5e12b26dd01dbbd9d3e42d35
d9de271302e0f1f69d87cbc7aca9e8867e9d428d3cab0666eb490d5fbab30bfff3f785d03f2072a43
bb9b5e54656a592cb61eafd5a5ef284c7caec66f7f47325cc0d4c1d27f661d8a748ca5071c06ef13
4dff96f4086688366d468a24780017e0b56aba7fab43b3b7c0b77906fae5482f32811c292e6b1445
4e14b894801a86a03cc47794dd0d74527a72e424ed3afa04899ecb9a63f2a9ae72be7fa989adf0d6
5a32c851d9801fc41048df33564fc7b31707ec8fb80140fe7b7a1fa120ba1cb660324cefffb4bcc2d
9bb7de0cf54c819f2dd3bceadec9c25f5e19dc9b10203010001</td></tr><tr><td>0.0.10</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009098865
def2f2ab376c7f0f738c1d87a27ac01afd008620c35cb6ebfcb0c33003193a388c346d302317270
12193bb76fd3004b864312c689ef5213cbb901101509deab94f26a732e637929da4c4cb32517e3ad
bb3811d50ac4c77c1fce8b651606215f34707f3e7265545e58c894609e28376bdb7775fe30439e0e
1592fdcb0c3ee1c305773d072a6b8957eafce1a11be965edaff3843366cb6a44ec25a890106e6247
567f76b550fda482baec6307d698ec88841fd66f23f210e47b8a9dcba6ba4e1fa716db33c80e3081
9496dcb5e5609fb6e7c615379badded427e9231b9254c2baf943608a86d698ae9a3c8639df887d6f6
b5a71385d24338d911a212bf71f1e2acc8b186b96ec8e69c86b6d058217776a09c9c68953edb5916
578b5a263b2f469e3b0c07eada71a447eea7f8fc1bb8074255567b7f0bd1e6afb0358718c98b429e
24b2298596fc76cf6af396ca9434d7926ec7d37d4b92af56d45feff8196095224a916c1ffe6b667e
255fc3ac8cccef920dc044b25003132b87806742f0203010001</td></tr><tr><td>0.0.11</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009bdd8e8
4fadaa3532fc4ce01a8a17d4c3b232f50a9790e262684edc4823e815a1bd5b20ecea7bf56e29f6bb
7b831fb3bf6efcd1475f0b8ed5fffb0b1385b96d166b629f0396a8fef5f06e4bca25ee4a1340ee263
a4d9bb020d8f472306f3d886138de7a019e059bd0afc902ccba1a213ae2daa60c8a013755fe0a48e
034f5b4023a2dadeaa88c54868353ac7a7a3df12b2fb6418774e9b14be6eab8cc27b88012ad6162d
a74e0eeb16135905f437374dab8586d750a26bbd3ac24aed878c4d53e651072c871e94d7acc575c9
67381734a53feaf4d7ba6bcdd241cc6458c6087d86302aa251c04f6d56b9c32d7d96624750ed0557
85d0773f43dc099b28c92281148e6c81f297ff9d166e000ac04b3124186775fcef75f5eba0c1032b
f130df6cd7a46211d0df3e0584d92ea67349d8490508eb4ef88f54c8c3d486de8719f10fa96feb85
cc796076ca781318ee2d9ed903ca1336040c59ad91a4d2f698e9108ae0edb9b1cb95ad33b197ffb1
8bd1ba8b56cbee2aae9585ece208a1e14b48564630203010001</td></tr><tr><td>0.0.12</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810090259f4
e3d9f0f394256548e9c7308b10b73403cc9094d97ad151b7706170b9772ceb64d662ecf901a8d7d
15d319a59c8b71071accd895b7c93610dc6976f67c4e1729ba8373ab7e52a3f3c8f265491dde69d6
e0999470e7445981131bd96c36e6865203fb2ebd5d50eadafb726396dec1d9174898b4e9be04c74d
304feadd9cbd3234c3b7f3306c99cb0c339fc25969b41d58a2b7cfc1832e226d81c1963993e2255a
087d1698c03d4210bd64580644d095ca76aa1794edd40c1c87b5f82a8e39f603e97116ba04578e7e
80346495d785d4ef7cf7714b9eb6f5f9e0b9a94f4b73884619b9274d4a95ef15754a89d97ef5c1a8
8b6d693e0a80ebd537fc9cf0ca91d1c62d915de7ed818b952e64c200293ee8e284a416a72a3e12fc
7d423b158f9b49660cbc2466fbed0fed2e24e102fde942eb4cfd94bec46d3d90fc08c39fecba03e0
ca2464ae664b979515ba29e1f702c3fe702be793796d8edb17aa48c09290b024549f0611f5ae23ed
7e16442df7d1dad2286c2bb09d5522dd3ed698c2f0203010001</td></tr><tr><td>0.0.13</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810082de730
65f34ffc29340d5949d2220b1e4366ed5cf7c6ebd616cf9416a53ea0017f6bb116bfd3f3defcc15b
7a4ddf0e44d02fe695688053e79a770e201bcf7193390039ee8f086d4fa746c7e056918301f9b5e8
4e39262828085a79b322bca0b5d85fe97221a26bbde258c620f0dcea02ab1edd16cc49a3f2ab9288
e3dd1f37dc4b6a6f7133ff92e541c71b70d2a2f66d55725ab18bf86d009ec3d24f5d12e0b5e6802d
1151372d4b764ebecb4af82f649485ec57b5a01dc67958f5a03ccaab7cba9354a17372c1316ba47c
953aaf94901b3f8c24e6a3afd6758e7f3b143ce2dd3cb071b2a74c921cee949a4b5a6be879f1c790
a6b8d63b192d7ee29a9491fdd689a98c0a7c3d60320f1b4ac2d6229dfd94e42f3a6048a76be1eb95
8c8a1873be8d338aec9fc59ab7f37626789402c1fd595f19087575e0be827fc4c0a4fb3d393ad74a
949cc986bfb64cabddae53935f6dc56074db93d77ea3b816bdd6be534497272289859ff34ce51860

affb621d10487dc3843f1f86d54034a63e48a1a0d0203010001</td></tr><tr><td>0.0.14</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810098755a408b5321e263052000d6d7d4a2c3a554d5e1384a9cb5ebf474ae882c63b486bd08d144ddf1a94ce9a7d6251963006afdaac458846f17640195fe2539a656930efa854f2148e68ec1a08c1c49d200c3f3045fe7147f06d534c4bd262100cb1dd39739d760d81a0bd20f83f255d2507d4ccb1106b53618c6a94409c887cae262d4cee9c86232147cec1404e0c57bba7317130ee39643888af3d598edd82b8c61e65ae81a4e1a56bc06d397143a98d41ca87d3ef433ef0aeab6801191b3e38480968f66b6e88662af45a9e212994f68b288eb967beb98478c243e2136c1a1591f061f5bc04b21ff2ba48b29f18431088873bdf99f8a52e9408971856e804dea602a311786c985652963c3a3770329b409f74fdcf746b22a5f8418912071c4ce846c9b4b320fedf6e9b64e2cbe384f9a82b6aaad4b20907431df1a33f69207a565600be81070d0832900995859a4498d5b59315bcebefee807eb0a3a942f1cdf3367dd4444fdb29886efcdd0be4abe9a188803953875eda33db72989f763b0203010001</td></tr><tr><td>0.0.15</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a9db7f8baa126898fab789115a3b5d89744f197e28041ae098f3e886c69871721e11bb0ad11f3ce9124aa961d6a0dc845f49765c3fab19958402676f564462bf281dba5588780f03e905798e184269aaa60f7a1472331e2fb1deadd877c84cbcb641ca9e5c8ad6e45bc159cb079fcb0d449cdcd8d9239c1a047e7b448da0cdca26610a25f296d96e7469b676d4a444516e7a59e85293a8086f840c052854e02a8cb2002dad35825be4d83b52fa91e8c73ff049746148862787c1118f924d31cbac1b44feff22d436b3979eadf9b43a4bfa72e15b4755fcab260e06a279c3bb73bc7f16a060d4d522fd490580388aa595d8044736e522f6424915f7803b7583e095cdf78c32519697de81b89fb50054753b1a17f9aafb064d84c992f9ab11ccbc8cb10814dcdf5264aa45f21bdefac82ccacaaf358e31373ee1ba4e7402fd8a70ea0c28ca5cc74dc42510c969cd2c459b1ec3688a01ea39a992710cd2297c98a84b6348a577804fdc234d3fe1903e2c21e172da28b59ae6e4c7e8edd8b71c49d70203010001</td></tr><tr><td>0.0.16</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a8ceac367eb1f1de5f0d9ef3eaf0df9b98448fe20808476562a060c51c289770b4cacfe92cb65569823e962c2a2c9fed53bd36ca3a122de1c525a582f25a4d7d628c1a3d5bdb8936aece7510e7554ee7033025c092c828eeb5738be02ed963da81a59205634ce9454577ab82f40f13f1ee55e0ae727e23c30284b1f44b99ace4ddc5f9ac7ad88d9fa2255935b24dcba8400642e16cf2532c0b0d6892904608715c4076f46d84a0e0fed36e76ccdc96355e7a26160945c2b54ae26cc00fd082326346eeeee7dd75f91911e99dbcb99ea4ac6ba056c33228d881d85831d9cc879593da1746dd0ee95dc2b96fe93bafcff2cd7d92958d78df33f205d7115ed9fac4db6f4cc60e56a5441da5b5b55fa5999902e958a6b6c44d810ddc56181241b87f22f059a6880e8021736d01897db65449ce817a2375d03551cb0de507c609a0c8030ecf4bfdeb213c03daa764a1821b724334f71f768d7aecb277052a7033765f07218056c78f2a87af18386d8f61a5cfc3f2ba4dd59915f13d38634d16957570203010001</td></tr><tr><td>0.0.17</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100af0b914257bf7a4655c4a50d0cad5e0a1e4581ed6623f0e8730f796b8f29c58178bcc6932c1fc31f39ef44b82d3c43b398737373fecb1295228a0fd50a14f3646d84fe1f467caeb98d463e2975e995b8d2e1e39f3bf6addc25ae35d65d02608e0345537966e2abce49b814bead3c1b757174ae30c00b0c43e99b80496b72d3c131f1c6e4fcd05f28117ef9e28c4303be4d8c7e042d58b83cc121945a2c65e7962caa9185938f3757df7cca95cf02b5e31944a3a619a0ac3f1e34b9b013d4c224c4f1e70fd9fd36983ef86ade518362cc8322c0f7b61a9ac75fb82e7b86d68bc0f099a09a14cac5a1d8d38f9a8a70cc37ff5cc3bbd2742fffd146255c171e6a178083271dce0fde681ed492cb59b0796d270175838dc5908107e3a6ea3f9a406b3d1130ccce3b4791e49bbc231603b46ab2d0f93d43be75ab9a4d710ea940e285a7b153b0ca7cddee6d9dce0ad8350c41d90c215b9588515afa0ac3365ae07e81f3bbb36bdbeac4b31bcb1aa4e82565b977f9dad85d626eef9aaa9ef8d7e3fb0203010001</td></tr><tr><td>0.0.18</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181008c07be305ad60b90ba2dab39b0ee7760e1a22f857522540d70b03b3f9e4875a3a29ab08088f144f57eb252e46ba59385d0e6d4270117da0abc1b3b80694c9a5058b86d61dfa06e716709c88e8feac7c3a0e1d25fc0aebf6a8f76fcb99f845fe181461cab6858b97c3a4027fb3712b14e6c0789de17d41764577e511417eb162692eb07ae1e7355235e9bb439047b6c01613782e7dd6f604daa4674661d53961f46c3faa6b7e76762d373b5b542b79ea963efbf33ac68198bb2b661cff676916ef372ad4c26c216c4bc4787c84ec32d184d77c75186c09cf3d9f91433ca9853119bab31fa6ad26f453e596d9bdeca68a5769bc8fee7a535d80c8c6f3efb1dfb288ab6a979854b7ce83124ec0d102aff94c3b74f9c378958c25eb933d1d53c1e805a18654d6d9186990f6570429f960f34e8b4f7fd9972dcbfe9240e074da2d355a5f7ef9c1af62ef5982a8174578b9c15c49ec566bdac30ccfcef09cdf708ad487424e9c1be653f9ee7660e7d942c1efa5da286e1addab06a9a33f9de946795b0203010001</td></tr><tr><td>0.0.19</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100be17c99647cec65a44407b535856b3c3baef5b54f565af58b8456ba8c7ce535d5ac72c61c44c76b3c57c8e8

64841637be10a83cfe39c092476d0dbe4d6cdcdcd720a30b5bfeb51a01a18f582c45f6c86993fcf7
df182935de1d86906044dcf35186935d9bd7eea7952352bebb4ef9ae0f7661e70a4237afa9899668
7ca48fcfc5b00d3807f054be0fa8c3bfa425038be6ef295164f22f73b7e88c94ea9be8aa4f3a245c
89b9d1fd5192f7a50b958b2ef8104b36f1bf8fd2cfb28c1421800c1c47e4ef98af150070cc6d69d1
7e8eb92f18a6aa1a65266a495238d103f8f695b57ecf373650a052008745721bea815627967c8076
365df8c4c7a7d4dd8f2c3850c18fba71eb60e6e8dfbd196e0537fd70b344ecbcc530dfc83da6fedf
49d51a90419502ba9d70cd35f1cf3c0694e2354f9064fdbf535eb23c27c0a43d0b78c1f867c61d98
695d8def7bc2a10bb6674c22f66aab0a91813dddf27cdb852c59ef79e1b9e1a075fa6ee27a7e3774d
bf4b26465427e6d5ab91fe7f0f3a71784eca182b50203010001</td></tr><tr><td>0.0.20</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a5ad2b7
643a04c055d2f8cd2511b15139fc45575621388e49c119b2f398aca110f61396b0c866de5063522b
b8540273e13f6d94ce1e60438f6afb00aaa64612f7145e9bce8bc1a53b941913aa76c9f3a2833fad
7cf285c7ac2d37f99f3c2cdb49de4d151e61678564f281f541424b41fa7c51b2a9602283c7d32ee0
0eb838da15c38afc96e061d97ced22165ff1aa959f1c4275b2d098c40586a5579fbb3cb90072704
120a8a66a5270f4fcfd1086c923690a35e7fd445e33ac03f139c6868556570cdc4aaf22107a6c1a4
42456a7c6c79ee04090e7e5d4f66bca60ca1f47b6dfb543dac3cbf19a7719a8f55b6f83b4a3b8a66
d60256d0a46551fa7024bd05631b8a5580877254c2f2f268cdc33d2dbbcfb733e9fbe233bb9cb59a
b31a0148b23e8c42680ff10af4c79a4d08346fb79a93d9629548eaf1bb124698faefa4cdd72442c0
3a04b733432f748903a325c283d456ab9ae921ae7ed3391e5d1787efdc23540a7b85c691ae870a07
f90b11c13b32ce43eae15b369685ce49177cc9850203010001</td></tr><tr><td>0.0.21</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181008d45c21
c0c95ef65a029d52c957fd0f85f20123da034e61671dde5475f07382a66c66cb4dc50504ddfd375
81083df8d1757730ed8d6f364df4c36a2651591955da201a2407fa8ab9b2313811225a0da230fbe3
80e090aa56efa4f202ec9b4823f6501d96ac698ebf26aacf3ee2d1f32a721c947e1076cf35b373da
1d87a36a152e00e71011792282e825ff171c5833b88570bfc6da8449e6f95f8b1265ab5551940315
53d1d576f93c42c0ca60aabac4c8dd162d8114f2b21511583c72539fe56c499a929de3a40a0d45c1
7c589c2d7988ce26eafc92a3d37b7ea0042d43e03afa6271b26255a6cccfae5371821d81e0b05c25
0b59f0a90741a0e0e88a09ed56c5b9780d095f0906f0b81d51263982aae01136c072d844a11d6da4
b2a61c644e1ab17f16ff48ee23fede8452f1e42e2d30a0790c25d42060e1d44a671a2eb23d114f68
c71e33f176db58a68b430054bc1d2983a23a32ea6ff95fa7c4d8e380eb296e98b7968ecf8454d817
c737eea5dd921eb86c16c7b29304a4a7ecbe5a3a10203010001</td></tr><tr><td>0.0.22</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100b05abe2
ab00fdd06c955e86710b0e06f1a92624a48ad1cbc8dfc6f2212962b0c30fdbd284a37c5a37658b63
c36ea8162561a8e4f946cbe5722c028801f0f281c70f8d88c7c00a2f2e29f597b799869ed8356df5
7c47be9944a2aaff650f9b4bba0dbc53dc880fdbb69ea451905d2802202f8e29c04a76d27af2eb7c
548485bf3f4694c90c41810888843792848835f7816707d3e8d76f4e67f5780bcf08813c55ec639a
9bd624178f5eb147d500af351e9ef1b1e342484ca260db7ccbae486f13cf265b5b1ab68806600805
3b20c3dedce771c9a08a0320aa9ce451eb9d983a7b49caa1096f8adc098318dc38e0e7cef0d8e5d5
57a0675685a1c9e256a2bc9dba322b3bb3172cf714077bc380f8a0a433a8bfa7fbfc59f6b093ec8b
f6e9397c09b18e18040c1b566864737c8fa7e29795f3a4588dda7c2bab495665cc4a9b836e2eb90c
62a3fcaf591fb5f81804c76180e626fa2644a7de34511d6c4667d98937e27733f4d1e913883354e5
4fd7351721e76f7b56c3483388f4a6b87b28aeb0203010001</td></tr><tr><td>0.0.23</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810098ed6b2
213a3da894b8e83c4532589097d1f9456c70b9fe2d8c308def36daa87706d430f23ca53b27d034de
ff5c2e0ac351a0729b4760a19b1525b9d20a1961b962255c4bb2a5cc05a05d7b476f6e5b0547b4a8
83b50f7d1c93745ba40366608106dbf05b755ebc51d1b8291d107f5c0d9a2483ebce3d07c8b7b585
7d62b4be56375cf2314f7e009e4f19e8c8089ec69f96d0266199cf7ea3363b157ae952b2283a8d9f
7ceb45738b152486f54d40f6200b7ea755d336e1c33ad58ffe03a8c5650ab62b93b2b6645769fe01
f4d21cfaa4ce99510f771de22ba9ad1e8c5ecd3404cc3183174c7c84fbdf108b49733868e7c9fecb
90b3bb85b0c3c1378032f3b798e6fb9f7fd35fd25f3c21e7cfae81a01bba50bc4fdf82222a1686e9
200a1b323b618e448990e2aefb30619048be3599ffc8880ede3b67ecc8bbd4df6432f521bfb337bf
2b0958e29e66223ae35dc09406be0212132be29581694a7402604f1ce689c4b57a5bafbc1d46b342
b51c31ff2b5675c6c1df60d128d4a64c66fb4f1830203010001</td></tr><tr><td>0.0.24</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009dcd8c0
a53e90c3559574f66204117d3b503e50a36d3097fac8429e6cecd37bb54071808f2ee982035f851a
0c9be2176383a22e38c1aba168f32f90570cb3233cfe625987666af67b514caef21fb8df6d0fcd33
cf2606b92ddea5536b6068d86782e39bd5c38445991d419b7d1ec08599412c0949d1c240b35c14dc
55274dba71ffae936125a5f819f54132e2439d4ac5597996ece85e13dff3361f9131f56ceac5b9f5
52b49cf6f9a9ac6e5dce2db369462f93af80e5b56b6e8bfa162a061b4a76892bdc84647306c6008

58fdd2703276c2c70440198efd7fe3545cf2ab580c74cfd6445aaf7bd7f745cc252eabd265eabee8
62417104e6948a55756fdc222df0a101524de1c3c08ccf043011ec7fe964edd8451a130147c07363
a35f11fdeef8f2a2b761757b4358ff89b75a48d67bdc6090693e0bb8679ecbb93ffdb3f3ed96bec9
3ef4656e3716ab87ce46ca8e1259c8fedde8f2f1ea0f3eb2c48e96551de12330345725f45ed69c85
75b51683afa472621826db22bb2d1c4f1e36464a90203010001</td></tr><tr><td>0.0.25</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100ae7af9c
608c1bdb289fb817ffc6c9577ed6d4f6d57f274b215f1390c7e79e490f9dcb74cffe0e8757d67f90
f4b20dd4451b7ff6631a6b45c9e403be5a66224955583d1aa4238ba6f946b71cca8c16cc7888b2f0
d333c635b8e5478fac3ba3f81457a1a16b04e2
b2252651b6c4688c5510d2edc21b03e0a9b283041beef6783c0089845ecc96e6d235c56685d248f8
391fe0afc8ee03f3b498696c6d9decf2fc990c219ef6d2cbb2a69a26528e6009632d82ead3a583
ef0c37c2b79068118e0320b9fcb318f5ae48c0877955d0831bc9521aaa88b93419fed9f462f900fc
42601056e24ce449eddbc849bb782c09a1a36ad5e4d4b5d7466b7763913f794b2771b07afb617444
fb6b4b7484d64e191b513fc8e2501043f725421cd57b073bed21b00314185d6887fbc2b548d90bb2
a3c9184ae944c326db8f37a7356aa882bad4c7947a80e16d0f02e382d5771f1987c1b76e88cde1cd
f2d1a92215ec68d9b204e80b5dc4675ff3aabf223f7787eb2415df5e389cff60fc40ca252000b476
8410203010001</td></tr><tr><td>0.0.26</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100b728010
58ccefb52139ba9c6868a9495e37ba6571c5bc0fb7f2b0d079005dd5ce12af705dfc5ace7fe4b011
e6497e1b065e5ef3d937553b1c7c10545ac5c91354ecfad6a6476468883cc66a4a5eb0e1c1d1bbf1
030d9650f67b075909e53696ea31acad5240111f75537b14e6eef6451d7b334d1804a2a5789b5e4a
c6bfa27955e5517851601b67cfd0bd2869c7ed1f4ce098e367f133b1958743c13c6f69011a8c475b
2126d3bc352387989c866fe219a16291e44a92d5471f6fc16862ec7bfe0cad5a4eea1c60829876fb
5a012e1dbc51c0c082d890b9635267dd99b4c81115826e816e22e336834c9303c533e202cfa6a191
392601ceedba3f9ca4de6117775e0fc6104341e73521139d76c3364de28af58f1473a08472c60916
a641ae97ddb9bb072832dfe5b92271fd47ef89c9828a0c7175418d1c9ea3202e2ba588130bbf2e38
4eb847c9ac2d6807a92e4ba133d5715f36266617f8da56c9c8b394bc002575ca2020cda9f9297a7d
045d251025817559e2ea52f54428328ec3db312710203010001</td></tr><tr><td>0.0.27</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100ca795b1
66d8343bed7ee146c955dd21b828e83078b2a6d314b677610fffd73eba291183a69b558e93c4e0e6c
68a2535b13aaf61f9a56e2901f7e1bb999fe178f4c9354f456fcd5cc8c98d259028afdd2fb1df666
22d343a0d51317677fee1eb7491693334a0f66b89665be40a4aeb376acc0ac0283c5f5d304b10306
96bfd6c073063f21fdfe7b9a2c839abf637828303bf2dea28da015920458f9fd94eac62fd174e37e
e08a43b7930b22214a5664deca8d348295e0c399aa8b8460821fb644d77024267ec942edf2dd0c5d
ab272640431333885f442e73cc194148f728c5a7c9b5a2966d3e3b25f983600fda0ea3a4be97cf7e
8f3b0fb648f0b35fa13d272ae7d4342a929b597b40d1c544439cadc26db99b545cae60741888313
5e9bf160f9779f58d08d502f5eb951bf1e10917c8039abdf09ac3614ede498839f3da27600168405
41fefc84ac93c7efd91852a44016bfdc68e38e50c3ae3cc545c37d83a06742aa8ae531d5b16be32c
6bdf06dcc2373013892f92e3ae53641d76cd2a8dd0203010001</td></tr><tr><td>0.0.28</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009f661e1
b6526bd231239280555db14a909404cdcc9188642b6d47cb2f3e9c916632e26a4504cff08ba70d14
6db317a08cc08200419b1b96945984c19cf38d61cb4ae6f11a6aa507eabc12a7291731363790816d
f3a65fb9f55b20218429c6d24cd19302b12bdb659ad6bb8171ed6f68d9e96dde1aa0af45c69c3bec
061c5615d81600bd2bd710d8135a500e37eb748fd296f7c568c26c0dcbe6afe8a2fc2bc3a7e53052
4c835128d76aa54b413efa035269fd6881497d5ab300ab4efda455f42a80ef39c6bc41931f46b891
299040b6d0f3de893f68254be0841aae4f5112097e09db8fffc80c8523c79ea81c6c1adecdd04115af
9dbbcbcb35a02dbe89db88eff6ffff8cd6b1f335ef4fa720008fa0b2a5b14022193eb9e2cb7f0372fa
9b7e881438ef7460fe1780ec653f2a3ce789c0f5cb6f7b2716148a2e1cb3edc1c6daa1e10a4c0b86
741b1bfc8c4d2f35cbb0b0bc14cffbd677512fc15b0df93f76b22c32437c2751c8329cddd8a56eca
7520072ef3d01acefc17bb38aa4d442bb56d3b06d0203010001</td></tr><tr><td>0.0.29</td><td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009b18967
c838877f85a6471ce9f164cef939b01830b9eb22c02cc6b72a907020b3e4c014dc711ea8e095b88d
0b4858ec86b05a8c3a59ac12d2b9fabccac282ceb618db00eb59716611d6706c81aa32d9dddc6a6c7
b396ba202fedeb33f289a887284ebfc07d166d02c2c6ed32c7324c3ec8ae22112854e18ab5ea07a6
15c5ef8004ec68ac70dc03003a47f7efe103edce257d28e7961f428f1cfa2e6cf71bf45c564b82cc
bda14a183f30c2c3d5a7afba7a004079e87702c249e96a7b2fdd562fc16759efe75abe6a23d0d2f9
06a2df1d4b64cb2117a7304449c75319a7620c219a4fffc982e822b6e1a07be1cf98be9265d086dc2
71aa406310f8a846fd331239fe303bd5616c89080fb88639b7c0ceb14009381823e0433db6f9156e
2bda1873d4aa9a3a639604bfbfd11a6dd6ce03b4b0ceef95601c7d88a840397cdbca3ff214fbf58c9
d9dbd79d39ea767e9ae5f6eab9fca05fc4800f557657c90c12c60e02164825d4c33af4737374ea23

5b50313ed0b75bf89a6b79001fba74cc1319e55150203010001</td></tr><tr><td>0.0.30</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100a13e000c6349b6b805ef58d0661201107b7546a59d2bd4e0d76333c384e3a33a9466c86fe78f5db80bafcab c177cf7c9abe62ac4883e9636975c6e95897854ea1cd3fb807b7e60be9eb1f285c90fad206be6fa0 1ca14cff11a2bdae96f00008baab443a53874c2593c76b0a3bddb4a20ad57b2f5fe444059e8591f6 b53bbf92396436f0a739423da834e74c3fcff32f4d697434088ccee7773172640dcb49422ff668a3 4311f70788497afb0dce61d9e3b6d4b83d53b31f27afd5c76ce77c9ef80bd7dfa928a74f1a1d239 47f2f1497826fa6f88044ee171858802134d9e2dae75b7e397aec3a826f890d700a8a2396fef9673 f77ae0633349410d9758747ea8ff2878fa73ac1dea79b7ce680f18fb9b80815a75be12413d56bad7 30116e0ea6e9038c2d247ef0ef95b1dd03f62b8f1a1366558897e4324ad06aa93fcd9516b64042f7 be40878ac78e7a9c40447fe38878f8218a52fe64132b39936e660309b04d07dc722fc495bc9bce81 38d0e62c86ec53c7f9cc997c3d6cd8a91520189eb0203010001</td></tr><tr><td>0.0.31</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810098e8c59 b2cc138b66c38eb8170d5f3d55aadbc07c85ff94c8e9ebf12a9476ae546eec946e5aac467592fae2 cbd3c3b0cfeeb6836e7ab7c1778036a7e2e1a67c27e2fa715552f22b87f6206b315098a9802840a8 754000c051e94e4033de37b847a1e1b6d8c5acace81dc7c4959b97508642bd7c32024fb1cdad7455 6b88d577e7a32e5c4703b49151a71cfb7010a14843f29cdc896ccaef2ee2a616910f22dd65f21d99 ac351525a0d246c066f4efb9bd4a64031ef1d2fc3f864ac8c8f814bb3ec00759697104f3b8b1f3d0 aade613c748f8422db60f5faa96f7ed9e66dd80d9b6d1edfddcadd791cef060e393082a61fea0b0f 996dd54e287f75bde01e6fcdd82698ccb90e3ee2e23596a8706bdd04f56fc94fc417061128536110 dd9c5825e72ee9c52a57b332edf6eb1fef1d3425fededad401bdb8c9e2a0706787ce78c00a7400739 842b03c57b4831ef661351d6282e4a6f2c248a7bf053d6d949665a682091465a4d0df42d81f54487 4f174976da717047ef606ce83240cf546fffb7a690203010001</td></tr><tr><td>0.0.32</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c7f0ccd 1d614e3b6ba9756f7f948200a14cb53f9c3348b741e3dd0d232d1513beb1fb6951e387bd5bfd95ed 3e9a01f490661630a0add78a13a1a229e3c60b28e806f94de72fe4576431215f0fdc1ad97f808c6d a3e9a0adc415973ec48b895664cd452af20daac42322b07ed35a61cb2700cdb698a148ecfbc65795 60904366a5544f965ce50fe98246fd6df2575989edc6185ec39ad22b9523fb1d2a116940da60adc0 d55d00315effe210d1bfb5b556b015a835c128d4a33d5f6938ccb73c1ef2f120e6bf00977f7e4eb3 174a4744e2215ad48737a096bd39e32fd99d530ce42a7bae55abe73b7d0a38ef9d0b4a9fd27523b6 c8d0dafe7161a2d28de82bcb74d751cb6c28341768c0ee065a48169637b924fa4c67c91db44caf57 5388904bdd3de40bd3d2955bac15acab97cac735ba4e7d0465437d4b7294de40324741f7c5f43348 9b5c0e5852c655508e1565945026f73c409fab58e3ac46b8f7666d54560954d58a4e87ee5b42354c 915e3273affbc1abdabe425eae9b79460bd7c4bf30203010001</td></tr><tr><td>0.0.33</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100922a6d7 61cda0bcdcd8139b1a67e2f6d42072b7b5289c5caf8ac340cee65d40fdc543abb60ddc7714824f14 669df6d42ebc44c7f68e18c5ac250eb5701ab08017e2ff5718b676fbf11efbc661f39f16829c5d31 b248a36b094629e03a6ddab4c925aef048706616cb53af193ac38e2c9a4e2730dbd1f75de816cc8 363867ebf6aaed5f02715526304b4523ac594f455a1692d3e7ec1bbe91267d172b8567de81e594e1 074c6faa9ed3028d0f46a8198e26e5f4d41abeb5f9ff8b734c9dde405f40ef79b4df3d4a3851bf36 05c729acfdded50d4f5bf5af37971d802790c1047c28c2c7567a5cf346b31ee0028d782369c2ce958 f6be9fffd8b1e96c8c61e0248dd38cd5b6adb367ba5fe3bec568683b5f79c159942cdd589a1705 97321756086a7cc2d003096fddd0ed245368a739687dcc55e2c3a024a3b573d26c8aedb465de699 8f6b29d7daf0fa16508d890979f46d74dc9fefee8cb0ffee7cd8bd9bd679b701df6cf7da2194425d daf4ee8040aeac65e60d58100fcd75381f68354630203010001</td></tr><tr><td>0.0.34</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009383257 eb68ac25c8ae76f77e545a995684f0a2496e24da29f5463c5a2e9562dd8b5626fe38566ff91fa7ee 311400f6b48e3e24943fdacb49c3318d6ff453e9bb89d591c5a92ae6fb99ce647b8bf4c4ec9532fb f505f0dcbc8962af1d39f9c757ddedbb062a086ebfcf8cb65d2176d5da74245d622cb46596aafd4b 21da4e08495c6274b2c5871097c0d9606e3d5c969edee0e2c24f03c2233db90db1d79d2e76fb8937 6853059de2a8c7b30c00c49e6a034b77edbe94291385f7181d13ef2e6eea6029210abb20838626e0 b50029bbb7837af32c588677068193d16609a57c964e488657a2ed1abfa75158877c2a65e3beb03e f5b1290f2a0161bf9830c14a095deb73ad062f33461feceeb046eac1a0dd36c98268b3306af9da66f 8fba97bd8b49be0065dfedd3ada7a8aaedf9d91c5553527662164fc6ec59455f65e199e574bd4850 98b962afd767766241c3d5aea1f545d51651848d4be1832ada53d93efea3a26ba4a9a94159f8f7f1 c3fc4cfe4b9dd846dd6bce976fc73084cb8cea0330203010001</td></tr></tbody></table>

description: Common questions for the current Hedera's permissioned Mainnet
consensus nodes

FAQ

What security protocols and key sizes are used by hashgraph?

The nodes use TLS 1.2 DH RSA 3k keys and SHA384 to secure communications amongst nodes. Our goal is to satisfy CNSA suite1 as specified by the US government. TLS 1.2 uses ephemeral AES 256 for perfect forward secrecy. Different keys are used for the TLS key exchange and a different one for signatures. Clients use ED25519 to sign transactions.

Does a consensus node support bonding or splitting ingress and egress traffic?

Hashgraph does not support bonding or splitting of ingress and egress traffic.

Does the consensus node need access to our internal network?

The Hedera consensus node does not need access to any internal resources and must be separated from the rest of the corporate network, ideally in its own DMZ (Demilitarized Zone).

Does the consensus node need to be backed up?

Application specific backups are not required. Since the Hedera network continues to process transactions while the failed node is down, restored backups will be out of date by the time they are recovered. Redundancy comes from the other nodes and the recovered node will be resynchronized by the hashgraph software.

It is expected that normal backup procedures are in place for the operating system level to allow for rapid and consistent recovery for disaster situations including hardware failures and similar situations.

What are the consensus node SLA and operational requirements?

The LLC agreement specifies that "while a desired initial target should be at least 99.5% availability, inclusive of scheduled technical/operational maintenance windows".

External monitoring will be available from Hedera for notification of failure.

README.md:

description: >-
The Hedera Mainnet is currently comprised of permissioned consensus nodes
operated by the Hedera Governing Council

Node Requirements

The following is provided to help Hedera Governing Council members deploy their permissioned mainnet consensus node. Please note, this information is not intended to apply to Hedera's transition to a permissionless network.

Minimum Node Platform Requirements

Currently, the Hedera Mainnet will perform at a rate determined by the lowest-performing node. To ensure a common level of performance minimum hardware,

connectivity, and hosting requirements have been defined for the initial permissioned, Governing Council nodes.

{% hint style="warning" %}

To ensure accurate conformity with the minimum requirements, please provide node hardware, connectivity, and hosting details to Hedera prior to purchase (devops@hashgraph.com).

{% endhint %}

CPU: X86/X64 compatible (Intel Xeon or AMD EPYC); 24 cores/48 threads meeting or exceeding the following benchmarks:

Geekbench 6 single-core score

Minimum: 1000 or greater

Recommended: 1500 or greater

Passmark single thread rating:

Minimum to remain on Mainnet: 2300 or greater

Recommended: 2800 or greater

Network Connectivity: Sustained 1Gb/s internet bandwidth via a single 1-Gigabit / 10-Gigabit Ethernet interface

Memory: 256 GB PC4-21300 2666MHz DDR4 ECC Registered DIMM or faster (minimum), 320GB or higher PC4-25600 3200MHz (recommended)

Storage: It is recommended to mount 240 GB SSD with Raid 1 as a root volume / and then provide usable storage via different devices later mounted during installation. This may not be possible on your hardware, so alternatively all required storage may be allocated to the root volume.

Minimum: 5TB of SSD NVMe usable storage

Recommended:

2 x 240GB SSD with RAID 1 for OS Storage

2 x NVMe devices as a 7.5TB RAID 0 (or 4x as RAID 10 array)

Storage performance: If mounted to root volume, the root volume must meet these requirements. If provisioned via RAID, the RAID array should meet these requirements:

Sequential write sustained:

Minimum: 2,000GB/s

Recommended: 3,000GB/s

Sequential read sustained:

Minimum: 3,000GB/s

Recommended: 4,500GB/s

Random read, synchronous:

Minimum: 250,000 IOPS

Recommended: 375,000 IOPS

Random read, AIO:

Minimum: 500,000 IOPS

Recommended: 750,000 IOPS

Random write, synchronous:

Minimum 100,000 IOPS

Recommended: 150,000 IOPS

Less than 200µs random read latency, average

Nodes must pass the Hedera performance test suite performed at installation time

Node Operating System:

Linux

Minimum kernel mainline versions (not distribution version)

6.2.0

6.1.2

6.0.16

5.15.86

Actively Supported Long-Term-Support (LTS) 64-bit Linux Distributions

Ubuntu LTS 22.04

Red Hat Enterprise Linux (RHEL) 8 and 9

Oracle Linux 8 and 9

Node Software:

- Docker Engine (docker-ce version 20.10.6)
 - Deployed with root privileges
 - Privileged container support enabled (optional)
 - If privileged container support is disabled then host machine must run the Havege Daemon
- Docker Compose (docker-compose version 1.29.2)
- IPTables Support (linux-kernel version 3.10+)
- Havege Daemon (haveged version 1.9.14)
 - If privileged container support is enabled then this requirement is optional
- HashDeep Utilities (hashdeep version 4.4)
 - Required for update integrity validation
- Bindplane Collector (bindplane-collector version 4+)
 - Required for node software log monitoring
- JQ CLI (jq version 1.5+)
 - Required dependency for the Node Management Tools
- GNU CoreUtils (coreutils version 8.00+)
 - Required dependency for the Node Management Tools
- cURL CLI (curl version 7.58.0+)
 - Required dependency for the Node Management Tools
- InCron Daemon (incron version 0.5.12+)
 - Required dependency for the Node Management Tools
 - Required for automated network upgrade
- Rsync CLI (rsync version 3.0.0+)
 - Required dependency for the Node Management Tools
 - Required for automated network upgrade
- Node Management Tools (node-mgmt-tools version 0.1.0+)
 - Updates deployed via the node upgrade process
 - Must be installed at the following path: /opt/hgcapp/node-mgmt-tools
 - The path must be writable and executable by the hgadmin user account

System User Accounts:

Node Software Account (mandatory)

User Specification

Name: hedera

Unix UID: 2000

Group Membership

Primary: hedera

Secondary: admin or wheel (depending on Linux distribution)

Permissions:

Read, Write, and Execute Access to the entire /opt/hgcapp folder tree

Group Specification

Name: hedera

Unix GID: 2000

{% hint style="info" %}

Note: Reference Configurations available in Appendices B, C, D

{% endhint %}

Proxy

Access to the node via public APIs must be mediated by an in-line proxy. Below are the specifications for establishing this proxy.

2-core x86/x64 CPU

2GB RAM

100GB SSD storage

200Mb/s sustained internet network connectivity with public static IP address

Supported Docker (Hedera to provide Docker image with HAProxy)

Network Connectivity

Node Connectivity

- 1Gbps internet connectivity – sustained (not burstable)
 - Unmetered preferred
 - Deployed with firewalled access to other mainnet consensus nodes
- Node deployed in dedicated (isolated) DMZ network
 - Static IP (FQDN is not supported)
 - TCP Port 50111 open to 0.0.0.0/0
 - TCP Port 50211 open to 0.0.0.0/0
 - TCP Port 50212 open to 0.0.0.0/0
 - TCP Port 80 open egress to 0.0.0.0/0 (for OS package repository connectivity)
 - TCP Port 443 open egress to 0.0.0.0/0 (for OS package repository connectivity)
 - UDP Port 123 open ingress and egress to 0.0.0.0/0 (for NTP pool synchronization of system time)

Proxy Connectivity

- Static IP address (FQDN not supported)
- 200Mb/s internet connectivity
 - TCP Port 80 open egress to 0.0.0.0/0 (for OS package repository connectivity)
 - TCP Port 443 open egress to 0.0.0.0/0 (for OS package repository connectivity)
 - TCP Port 50211 open to 0.0.0.0/0
 - TCP Port 50212 open to 0.0.0.0/0

Interface Bonding (optional)

If using interface bonding, note that mutual TLS is in use, and Layer 3 Policy Based Routing (PBR) with dual-pathways is not supported. Only Layer 2 interface bonding using mode 1 (autonomous ports using active-backup) or mode 4 (LACP 802.3ad active/active) is supported.

Hosting

- Industry-standard hosting requirements for security and availability
 - Tier 1 Data Center Hosting facility
 - SSAE 16 /18, SOC 2 Type 2 compliant
- Hedera will seek to avoid duplicating hosting providers across Council members

Network Topology /(Typical Corporate Datacenter Configuration/)

Deployment Steps

The following steps outline the process for Council Members to add their consensus node to the mainnet.

1. Initial contact with Council Member and node hosting entity
 1. Identify key individuals and project managers
 2. Establish regular deployment team meeting cadence
2. Conveyance of technical requirements and discussion of deployment options
3. Node platform acquisition
 1. Hardware or virtual instance
 2. Network connectivity
 3. Hosting facility
4. Configuration of the operating system on platform
 1. Provisioning of accounts as specified
 2. Provisioning of network access (firewall rules/access control lists)
5. Conveyance of credentials to Hedera
 1. Includes any special instructions for permissioned access such as VPNs
 2. Discussion of support and escalation paths between organizations
6. Hedera undertakes configuration review
 1. Platform

2. Connectivity
7. Deployment of Hedera consensus node software and required supporting libraries
8. Add connection configuration for a Hedera performance testnet
 1. Hedera executes functional, stability and performance tests for all network services
9. Review of test results and determination of preparedness for mainnet connectivity
 1. Review key management documentation related to Council Member's accounts including: fee account, proxy staking account, et al.
 2. Update private keys using provided tools
10. Schedule mainnet connection
11. Mainnet live

mirror-node.md:

description: Hedera mirror node release notes

Hedera Mirror Node

For the latest versions supported on each network, please visit the Hedera status page.

Latest Releases

v0.112.0

Our sharded database, Citus, saw some major performance improvements that should make it suitable for a replacement for our production instances. Transaction hash look-ups saw a speed boost due to the addition of a new distributionid column used to target the appropriate shard of data. A performance bottleneck caused by the use of SSL for communication between the coordinator and workers was identified and remediated. This change alone provides dramatic improvements across the board. The coordinator connection pool to the workers saw a boost as well, again improve overall performance. After seeing some memory problems due to the increase in query volume, we adjusted the workermem lower to a more appropriate level. Finally, we enabled topic message lookup in the REST API to improve the performance of the topic message endpoints.

This release continues the theme of improving test maintainability. There were a large number of tasks that refactored the web3 tests to use the web3j contract wrappers. This change will also help reduce the runtime of the tests and increase our security by not storing solidity binary files in source control. Similarly, a new RecordFileDownloaderPerformanceTest was added to provide a foundation for further performance testing. This class can bulk generate record, sidecar, and signature files using a declarative configuration and feed the data into the importer to test its performance. Such capability will be used in the future to perform 10K TPS performance testing in CI to improve our release velocity.

v0.111.0

This release updates the accounts REST API for HIP-540. Immutable keys will now return null instead of 0x3200 to match the behavior of the CryptoGetInfoQuery HAPI query.

Initial support for HIP-869 dynamic address book was added. The importer can now ingest the new NodeCreate, NodeUpdate, and NodeDelete HAPI transactions. In this phase, these transactions do not update the address book information the mirror node uses to verify stream files or the data returned via its APIs. The one exception is the new adminkey that is used to update the internal mirror node

state of the address book, but is not yet exposed via any API.

A number of important test improvements were completed. The web3 module continues to see its tests refactored to use a new object oriented wrapper classes to invoke its test contracts. This avoids boilerplate solidity function parameter encoding and makes it easier to debug tests that fail. With the recent swing in hbar price, it became more difficult to choose an appropriate starting balance in hbars for the acceptance test operator account. To solve this, acceptance tests now specify the starting balance in USD and use the exchange rate information to convert it to hbars so it is not impacted by market fluctuations. Finally, the RecordFileParserPerformanceTest was updated to support the 10,000 TPS canonical tests we conduct in the performance environment. This lays the groundwork to shift left in our testing and move performance testing into GitHub Actions.

v0.110.0

HIP-968 adds the ability to query for tokens by name. The `/api/v1/tokens` endpoint gains a new `name` query parameter that returns one or more tokens that match a subset of the token name. Searches are case-insensitive and can match any part of the name. Note that it cannot be combined with the `account.id` or `token.id` parameters and pagination is not supported.

HIP-1008 adds a `transaction.hash` query parameter to `/api/v1/contracts/results/logs`. This makes it easier for Ethereum tools like Hardhat and Metamask to retrieve transaction receipts for HTS operations.

HIP-869 dynamic address book continues in this release. The design was updated for the new `adminkey` field and support was added for the new `NodeCreate`, `NodeUpdate` and `NodeDelete` transactions.

Upgrading

This release requires the `pg\trgm` PostgreSQL extension to be installed to support querying for tokens by name. Failure to install the extension before upgrading the mirror node will cause the importer to fail on startup. First, verify with your SQL provider that it supports the `pgtrgm` extension. Then run the following SQL on the `mirrornode` database as the owner:

```
create extension if not exists pgtrgm;
```

v0.109.0

This release adds support for the `TokenReject` transaction from HIP-904 Frictionless Airdrops. Similarly, support for unlimited max automatic token associations from HIP-904 was added to the `/api/v1/contracts/call` API.

There were a number of other notable bug fixes and performance optimizations. Please check the full changelog below for further details.

v0.108.0

HIP-801 Add support for `debugtraceTransaction` RPC API saw its implementation completed this release. The new `/api/v1/contracts/results/{id}/opcodes` is now fully implemented and will re-execute the given contract transaction and return the executed opcodes. As noted in the last release, this API is not enabled on Hedera managed mirror nodes and is intended for local execution. The next release will add additional testing and refinements, but for the most part the HIP is complete.

HIP-869 Dynamic address book work was started this sprint with a design document added that lays out its impact on the mirror node.

There was a lot of effort put into automating the mirror node deployment process. A GitOps model is already utilized to help automate a lot of the rollout process, but in the case of multi-cluster environments each cluster would have to be manually updated. We know leverage GitHub repository dispatch to automate the PR creation to update the secondary cluster whenever the first cluster completes and its automated testing completes.

Citus saw further refinements in this release including work to optimize the topic message lookup migration to improve its runtime from weeks down to below an hour. The contract logs and contract result APIs were optimized to improve their performance on Citus. As a result, we felt comfortable enough to re-enable Citus in testnet and beginning work on the mainnet migration.

v0.107.0

There was a release issue with v0.107.0 so this v0.107.1 was created to workaround it. This release contains the initial work towards HIP-801 debugtraceTransaction API. HIP-801 adds a new `/api/v1/contracts/results/{transactionIdOrHash}/opcodes` REST API that can be used to debug previously executed transactions. This API works by re-executing the transaction on the mirror node using the state at the time it originally reached consensus and returning the executed opcodes. Since this is a slow and resource intensive API, it will be disabled by default. It is expected that developers run a local mirror node with a forked state to debug the transaction locally. This API is still under development and currently returns 501 Not Implemented status code.

HIP-874 is now fully complete with both acceptance and performance tests added and passing. This endpoint along with any other missing endpoints were added to our monitor API to improve our production monitoring capability.

Most of the effort in this release went into Citus database related fixes and improvements. The accounts list and NFT transaction history APIs performance were greatly improved under Citus. The topic message lookup migration was converted to run asynchronously to reduce the overall time to migrate from PostgreSQL. A cache was added for the contract runtime bytecode, improving the performance of `/api/v1/contracts/call` under both both databases. A fix was put in place for inconsistent token balances and fungible token supply being returned in the API. The performance of various historical contract calls were improved by the addition of caches and query optimizations. Citus connection management was revisited and tuned to improve the throughput for cross shard queries.

v0.106.0

The HIP-857 NFT Allowances REST API is now fully feature complete.

Our second new Java-based REST API, HIP-874 Topic Metadata API, introduces the new `/api/v1/topics/{topicId}` endpoint for retrieving the topic metadata described in the HIP; including the memo, submit key, and deleted fields as well as additional relevant entity information.

For web3 `/api/v1/contract/call`, in addition to the existing requests per second rate limiter, a maximum gas used per second limiter has been added. By default it is configured at the maximum supported value of 1,000,000,000 gas per second. This can be set to a smaller value via the `hedera.mirror.web3.throttle.gasPerSecond` property.

The REST API Redis cache introduced in 0.104.0 is now enabled by default.

Our sharded database, Citus, continues to make progress with this release incorporating additional enhancements to the PostgreSQL to Citus migration script as well as performance improvements around token transfers and contract

log insertion.

Upgrading

Refer to docs/configuration.md for the full spectrum of hedera.mirror.rest.redis. properties that may be used to tune for your specific environment. The uri property must be set appropriately to interact with your Redis service.

When using a managed Redis service from a cloud provider (e.g., Google Cloud Memory Store Redis), setting the parameters for the REST service programmatically via the new properties may not be supported, and manual configuration by other means may be necessary.

If the Redis cache for the REST API is not desired, simply set hedera.mirror.rest.redis.enabled to false.

The previous web3 requests per second rate limit property hedera.mirror.web3.evm.rateLimit has been renamed to hedera.mirror.web3.throttle.requestsPerSecond.

v0.105.0

A design document was added for the implementation of HIP-904 Friction-less Airdrops on mirror node. Please watch the epic to monitor the progress of airdrop development.

Citus, our sharded database, continues to make progress with this release making its way to one of our two testnet clusters. We'll monitor its deployment for a period and make any necessary fixes. The ZFS CSI driver we use for Citus saw an upgrade. Finally, multiples issues were fixed with the PostgreSQL to Citus migration script.

For /api/v1/contracts/call, the maximum data size was increased to 128 KiB to provide better Ethereum compatibility. Also, additional logic and validation was added to more closely align with consensus nodes error scenarios.

Upgrading

If you're using the ZFS CSI driver, please ensure its CRDs are updated before upgrading:

```
for crd in zfsbackups zfsnodes zfsrestores zfssnapshots zfsvolumes; do kubectl
patch crd $crd.zfs.openebs.io --type merge -p '{"metadata": {"annotations":
{"helm.sh/resource-policy": "keep", "meta.helm.sh/release-name": "mirror",
"meta.helm.sh/release-namespace": "common"}, "labels":
{"app.kubernetes.io/managed-by": "Helm"}}}'; done
```

v0.104.0

This release adds a Redis cache to the REST API to improve the performance of /api/v1/transactions. This functionality is currently disabled by default as we fine tune it.

HIP-857 NFT Allowances API made a lot of progress this release. It's taking a bit longer than a usual API since it is our first Java-based REST API that requires some extra groundwork. This release enables the rest-java Helm chart by default allowing users to test out the NFT allowances API in all environments. While most functionality is present, please be aware that some parts are still under development. A new index was added to the NFT allowance table to speed up look-ups by spender account. The existence check for numeric entity ID was removed to improve its performance and to better support partial mirror nodes.

Finally, we added initial acceptance tests to verify the API end to end.

Our Citus deployment is nearing the finish line. Citus is now deployed to previewnet and it now runs both PostgreSQL and Citus deployments in parallel. Internally, we've deployed it to a mainnet staging environment with a full size database for further testing. This deployment was possible due to the dramatic increase in migration time we implemented this release. Mainnet previously took more than a month to migrate but with this release it should complete within a week or so. Expect testnet to be migrated to Citus very soon as well.

v0.103.0

This release adds support for making metadata information from HIP-646, HIP-657, and HIP-765 available in the REST API. In particular, this adds a base64 encoded metadata field to the `/api/v1/tokens` endpoint. It also adds metadata and metadatakey fields to the `/api/v1/tokens/{id}` endpoint.

The contract call API saw some noticeable performance improvements with the implementation of lazy loading for nested items. Previously it was eagerly loading all the account information even for simpler calls that didn't need the data. With the switch to make these additional queries lazy, we see an improvement of 50-90% in request throughput. That change plus an improvement in the performance of the NFT count query should result in additional performance and stability of the API.

Work is still underway on HIP-857 NFT allowance REST API. This release adds EVM address and alias support to the new endpoint and fixes the error response format.

v0.102.0

This is a smaller bug fix release with incremental improvements to some in-flight projects.

For HIP-857, an alpha version of the NFT allowance REST API is now in place. It can be used to experiment with while we work towards implementing the remaining query parameters and squashing any bugs. The Jooq library was integrated into the rest-java module to allow for dynamic SQL querying based upon user input. The next release should leverage this functionality to fully implement the remaining parts of the API.

Our Citus implementation was successfully deployed to the performance environment and it is passing initial benchmarks. Preliminary results show that Citus improves ingest performance by 600ms while sharding the data across multiple nodes. Promtail was enabled on Citus nodes to capture database logs and a new ZFS dashboard was added to Grafana.

v0.101.0

This release adds support for storing the new mutable metadata information available in HIP-646, HIP-657, and HIP-765. For now, it just persists the data and in future releases we'll expose it via the REST APIs.

The `/api/v1/tokens` REST API now supports multiple `token.id` parameters. This allows users to efficiently query for multiple tokens in a single call.

The `/api/v1/contracts/call` REST API saw some major performance improvements this release. The first change was to switch the Kubernetes node pools to a different machine class that provides dedicated resource allocation. The endpoint was also switched from a reactive MVC stack to a synchronous MVC stack. Simultaneously, the module enabled the new virtual thread technology that replaces platform threads. These changes combined to improve the request throughput by 1-2x.

Another important change was to enable batching between the download and parser

threads in the importer. For now, this functionality is configured to behave as before with no batching. When configured manually, this can reduce sync times for historical data by at least 12x. In the future, we'll look at ways to automatically enable this functionality when the importer is attempting to catch up.

v0.100.0

This release implements HIP-859, adding support for returning gas consumed in the contract result REST APIs. The current gasUsed field in the API holds the amount of gas charged, while the new gasConsumed field holds the amount of gas actually used during EVM execution. Providing this extra data will allow users to provide a more accurate gas when invoking a contract and reduce the fees they are charged.

/api/v1/contracts/call now supports a configurable max gas limit property hedera.mirror.web3.evm.maxGas. The default value remains at 15 million but operators can now choose to increase it to suite their needs. The EVM version and features have been upgraded to v0.46. This brings feature parity with the latest consensus node software for EVM execution.

There was a large amount of work to improve our integration with Citus. Three repeatable migrations were enhanced to work optimally with Citus: account balance migration, token balance migration, and synthetic transfer approval migration. Token account insertion was optimized to improve its performance by removing the join on the token table. Range partitioning was removed for entity related tables since it caused degraded performance due to having sparse partitions. Finally, the deployment now supports different sized disks for individual workers to optimize for unbalanced data.

v0.99.0

This release contains the implementation of HIP-873 adding token decimals to the REST API. Previously users had to make $N + 1$ queries to determine accurate token balance information by querying /api/v1/accounts/{id}/tokens once and /api/v1/tokens/{id} N times to get the relevant decimal information. This HIP adds decimals to both /api/v1/tokens/{tokenId}/balances and /api/v1/accounts/{id}/tokens so that decimal information is directly returned alongside the token relationships and the additional N queries are unnecessary. It also adds name and decimals fields to the /api/v1/tokens response to expose more of the existing token information on that API.

The /api/v1/contracts/call REST API now supports a configurable hedera.mirror.web3.evm.maxDataSize property so that mirror node operators can adjust how large of a payload they wish to support. The default value for the max data size was increased from 24 KiB to 25 KiB for creates and from 6 KiB to 25 KiB for calls. This change makes it possible for view functions with large inputs like oracles to now work on the network.

There were a few items to improve the performance and security of the mirror node. A new hedera.mirror.importer.downloader.maxSize=50MiB property controls the maximum stream file size it will attempt to download. This protects the mirror node against large files uploaded accidentally or via malicious actors. The importer was refactored to support batch stream file ingestion so that it is possible to process multiple stream files in one transaction. This will help pave the way for future enhancements like improving historical synchronization times.

The database saw a number of improvements including new setup documentation with recommendations for how to configure the database. Our Citus deployment had some notable additions including a huge improvement in performance by adjusting its resource configuration. The Stackgres version was upgraded to 1.8 and ZFS to 2.4.1. The entity stake calculation was optimized for Citus so it runs efficiently in a sharded database. Finally, database metrics were fixed so that

partitioned tables are appropriately aggregated under the parent table name.

v0.98.0

This release saw the implementation of HIP-844 Handling and externalization improvements for account nonce updates. This HIP resolve issues where the consensus nodes and the mirror nodes are account nonces are out of sync. The consensus nodes now sends the mirror node the up-to-date account nonce instead of the mirror node attempting to increment the nonce based upon its prior state.

There were two important changes to the database that helped to reduce its size substantially. The topicmessage table primary key index was dropped in favor of relying upon a similar index on the transaction table. This simple change shaved 800 GiB off the mainnet database. The staking reward calculation performance was improved to only write accounts that elected to receive rewards. This reduces the staking reward calculation runtime from 47 minutes down to less than 2 minutes. A migration also removes the existing staking rows that did not have a staking reward election, shrinking those tables by 155 GiB. Note that to realize these disks savings mirror node operators will need to manually perform a full vacuum on the entitystake and entitystakehistory tables. So in total the size of the mirror node database was reduced by almost 1 TB this release!

There was quite a bit of technical debt paid down in this release. We've removed support for the event file format from the importer. This format was never fully implemented in the mirror node, didn't support the latest version, and no user interest in this data was expressed during its 4 years of existence. The acceptance tests were refactored to use the OpenAPI generated models, ensuring we dogfood our own API specification. The brittle MockPool tests were removed in favor of additional coverage in other, easier to maintain tests. The REST API tests now uses the correct read only user and common database setup that the other modules use. Finally, the unused RestoreClientIntegrationTest and associated test images were removed.

Our Citus deployment saw a number of improvements. Performance was optimized for hash insertions by reducing the shard count for hash tables. Entity upserts saw improvement by increasing the number of CPU resources to the database. Finally, the transactions list and accounts by ID endpoints saw their read performance improved for Citus.

\

v0.97.0

This release sees some incremental changes to the REST API. The REST API now supports a RFC5988 compliant Link header in its response as an alternative to the links.nextin the response body. Either link can now be used for pagination, but the Link header provides a standard approach that's supported by more tools. The /api/v1/accounts/{id}?timestamp endpoint now shows historically accurate staking information in its response so that users can view their past pending rewards. The timestamp in the response of the /api/v1/tokens/{id}/balances endpoint is now more accurate by reflecting the max balance timestamp of the tokens in its response.

The helm chart was verified to be compatible with Kubernetes 1.28 and saw its dependencies all bumped to the latest release. The new rest-java module was converted from WebFlux to servlets with virtual threads. This should increase its scalability once we implement HIP-857 NFT allowance REST API in a future relase. Some internal refactoring of BatchEntityListener to BatchPublisher will enable future improvements to historical syncing and batch processing.

The /api/v1/contracts/call endpoint saw a lot of important bug fixes this release. Support for historical blocks should be complete with some remaining bugs ironed out. The supported operations documentation was updated to reflect this increased level of compatibility.

v0.96.0

With a lot of the developers taking some time off for the holiday season, this release is a bit smaller than normal but still contains some important changes. The previewnet and testnet bootstrap address books were updated to reflect the current state of the network. The default volume size for Loki was increased from 100 GB to 250 GB to account for increasing amounts of log activity. The processing of EthereumTransaction was made more resilient so that the importer does not halt if encounters a badly encoded transaction. Finally, a memory leak in the REST API should greatly reduce out of memory errors and improve request throughput.

To improve ingest performance of entity tables when used with a distributed SQL database we introduced a new temporary database schema. This schema is used to hold the temporary data inserted when processing entities from a record file. Previously this information was added to temporary tables created within the transaction scope, but these temporary tables could not be made distributed in Citus. With the temporary schema, this information can now be distributed appropriately to ensure optimal ingest performance. This change does require manual DDL statements be ran before the next upgrade (see next section).

Breaking Changes

As previously mentioned, a new database schema was introduced to handle the processing of upsertable entities. This change\ doesn't require any manual steps for new operators that use one of our initialization scripts or helm charts to\ configure the database. However, existing operators upgrading to 0.96.0 or later are required to create the schema by\ configuring and executing a script before the upgrade.

```
PGHOST=127.0.0.1 ./init-temp-schema.sh
```

Another breaking change concerns operators using our hedera-mirror-common chart. The aforementioned Loki volume size increase was made to the embedded PersistentVolumeClaim on the Loki StatefulSet. Kubernetes does not allow changes to this immutable field so to workaround the StatefulSet will need to be manually deleted for the upgrade of the common chart.

v0.95.0

This release saw the Java components upgraded to use Java 21. In a future release, we will explore the new language features in 21 like virtual threads to unlock additional scalability. Some technical debt items were tackled including removing redundant test configuration by creating a common test hierarchy. Explicit @Autowired annotations on test constructors were removed, reducing boilerplate. Finally, various classes were renamed to align to our naming standards including the removal of the Mirror prefix from classes that were not used across modules.

HIP-584 EVM archive node for historical blocks saw some major additions including initial support for historical blocks. EVM Configuration is now loaded based upon block number instead of always utilizing the latest EVM. This ensures that /api/v1/contracts/call simulates the execution as it would've been on consensus nodes at that point in time. Database queries were adapted to work with timestamp filters to allow for returning historical block information.

Our distributed database effort saw some notable improvements including upgrading the version of Citus to 12.1. PostgreSQL 16 support was tested confirming compatibility with both regular PostgreSQL and Citus. Both /api/v1/topics/{id}/messages/{sequenceNumber} and /api/v1/topics/{id}/messages

saw optimizations implemented when used with Citus.

Upgrading

If you're compiling locally, ensure you have upgraded to Java 21 in your terminal and IDE. For MacOS, we recommend using SDKMAN! to manage Java versions so that upgrading is as simple as `sdk install java 21-tem`. If you're using a custom Dockerfile ensure it is also updated to Java 21. We recommend Eclipse Temurin as the base image for our Java components.

v0.94.1

Provides an important fix to pending reward calculation that regressed due to the balance deduplication work. The database migration for this will take approximately 17 minutes on mainnet.

v0.94.0

This release is mainly a bug fix release along with some minor technical debt items. A new Helm chart for the new REST Java module was added in anticipation of future work to support new APIs in Java only instead of the current JavaScript based approach. Support for Elasticsearch metrics export was removed in favor of relying solely upon Prometheus. The `/api/v1/contracts/call` API some notable bug fixes and performance improvements. Finally, some technical debt was tackled by refactoring `SqlEntityListener` to use a new `ParserContext` which should reduce its maintenance burden.

v0.93.0

This release deduplicates balance history resulting in a major reduction in database size with no loss in balance granularity. The mainnet database saw a 45% reduction going from 50 TB to 28TB! This deduplication process works by not updating balance history if the account did not experience a balance change since the last snapshot. A migration to deduplicate historical balances runs asynchronously in the background and against mainnet state took about 24 hours to complete. Because the index was changed to reverse the order from (timestamp, accountid) to (accountid, timestamp), this required a large effort to rework queries in multiple REST APIs. Also, the balance tables are now partitioned and this meant changes in our database metrics to properly aggregate child tables on their parent name.

HIP-584 continues to chug along with multiple bug fixes and optimizations. Changing per request objects to be singletons resulted in a large decrease in memory and CPU usage, allowing more concurrent requests to be handled. Web3 k6 tests were hooked into our automated performance testing to ensure they run every release to ensure no regressions. Finally, support for historical blocks made progress with more of the plumbing put in place to process non-latest blocks.

A new cluster database health check was added to the monitor to provide proper failover in multi-cluster deployments. The local file stream provider now allows for input files to be grouped by date for faster processing when the directory contains millions of files. This is a step towards a faster historical syncing mode. Finally, REST API queries were optimized for Citus deployments so that they could reach parity with PostgreSQL.

v0.92.0

HIP-584 Mirror EVM Archive node saw further refinements in this release. Memory usage was optimized by making most classes stateless and leveraging `ThreadLocal` where appropriate. Work continues on making `/api/v1/contracts/call` support historical blocks with lower level support for historical contract state, account, and contract information added. Additional test coverage for estimate gas was introduced to compare gas estimation is close to the actual gas used via

HAPI. Finally, an issue with the blockhash operation returning 0x0 was resolved.

We added an option to deduplicate account and token balance data for Citus that can dramatically reduce the size of the balance tables. Balance tables currently consume 50% of the database. In the next release, we will bring this capability to regular PostgreSQL to realize those cost savings there as well. We now no longer update the token history for supply change operations like mint, burn, or wipe thus reducing the amount of data in this table.

v0.91.0

This release adds support for ad-hoc transaction filters using the Spring Expression Language (SpEL). SpEL filters can be used for both including or excluding transactions from being persisted to the database. Previous filtering capability allowed mirror node operators to include or exclude certain entity IDs or transaction types, but if they needed something more custom they were out of luck. SpEL itself is pretty powerful and allows access to any bean or class on the classpath, so to reduce the attack surface we limit it to only allow filtering on the TransactionBody and the TransactionRecord contained within the record file. See the docs for additional details and examples. Below is an example that only includes transactions with certain memos:

```
hedera.mirror.importer.parser.include[0].expression=transactionBody.memo.contains("hedera")
```

Monitoring saw improvements this release with newly added metrics for the size of table and indexes on disk to help track the growing size of the database. Likewise, new cache metrics were implemented to monitor the cache hit rate and size. Dashboards were updated to visualize these new metrics.

HIP-584 EVM archive node saw support for block.prevrandaos implemented. Also, support for pending, safe, and finalized block types were added with all three being equivalent to latest since Hedera's blocks are always final. Work has started on historical support for /api/v1/contracts/call so that specific blocks in the past can be queried. This release saw the lower level queries for token balance implemented.

HIP-794 Sunset account balance saw a couple of remaining items completed. The balance timestamp that was added in v0.89.0 was put to use to provide more accurate timestamps for the accounts and balances REST API. Finally, the reconciliation job was disabled since it doesn't make sense to reconcile from balance data that mirror node generates itself.

v0.90.0

This release was mainly focused on testing and bug fixes. Acceptance tests saw a number of improvements including a reduction in overall costs by caching contract creation used in multiple tests. Acceptance tests were added to verify support for HIP-786 staking properties in network stake REST API. An issue with exchange rates varying in different environments was solved by querying the exchange rate REST API and using that to calculate fees to HAPI. The importer had an option added to fail on recoverable errors to find record stream issues earlier in the lifecycle.

v0.89.0

HIP-786 adds support for enriched staking metadata exports to the record stream for use by downstream systems. The mirror node now ingests the new maxstakereward, maxtotalreward, reservedstakingrewards, rewardbalancethreshold, and unreservedstakingrewardbalance fields and persists them to the database. The REST API has been updated to expose this data via /api/v1/network/stake.

HIP-794 Sunsetting Balance File saw further refinements in this release. The mirror node now captures the consensus timestamp at which the balance was updated for both accounts and token relationship. This information will be used in the future to provide more accurate balance timestamps in the API and to deduplicate the balance information. Entity balance tracking and migration was enabled in the Rosetta API. Finally, we now track the balance of all entity types and not just accounts and contracts.

HIP-584 Mirror EVM Archive node continues to improve with the addition of support for the PRNG system contract. Missing Besu internal precompiles for the Istanbul release are now properly registered. A lot of new tests were added in the form of integration, acceptance, and performance tests.

There were a number of technical debt items addressed in this release. The importer component saw noticeable improvements in CPU and memory usage at 10,000 transactions per second. It now uses about 50% less memory and 33% less CPU. The Log4j2 logging framework was replaced with Logback to provide a path to compiling to native code and to simplify configuration. The EntityId saw its final improvement with the addition of a cache to reduce allocating temporary immutable objects. Tests were standardized to use the simpler and logging framework agnostic OutputCaptureExtension. Finally, we researched approaches to parallel transaction insertion and saw a path forward for additional ingest scalability.

v0.88.0

This release contains support for HIP-794 Sunsetting Account Balance File. Consensus nodes will soon stop generating account balance files every 15 minutes due to the growing number of accounts making this operation unsustainable. To fill in the gaps, mirror nodes will now generate balance snapshot information from the record stream. This change will be transparent to end users and operators alike since the same data will be returned by the various APIs. For now, we're generating synthetic accountbalancefile rows (not files) until we can remove the reliance on this table everywhere. In this release, we updated the accounts by ID, balance, and network supply REST APIs to not depend upon this table. Entity stake calculation and a fungible token migration were updated similarly. The next release will see further work in this area.

HIP-584 saw the exchange rate precompiles tinycentsToTinybars and tinybarsToTinycents implemented. Also added was support for the HTS redirectForToken precompile. But the main focus was on testing and bug fixes with a large number of them squashed in this release.

There was good amount of technical debt addressed in 0.88. For starters, we have a new hedera-mirror-rest-java module that is intended to contain new or existing REST APIs converted from JavaScript. By creating any new REST APIs in Java and slowly converting existing APIs to Java we can improve the quality of this area of the codebase and promoting code reuse with the other modules. A community member helped us to remove the deprecated Spring Cloud Kubernetes property spring.cloud.kubernetes.enabled since it was no longer used anyway. We also took the time to remove unused Flyway placeholders properties and eliminate code redundancy in web3 acceptance tests. Finally, we removed the type field from the widely used EntityId in the codebase and eliminated the unnecessary AssessedCustomFeeWrapper.

v0.87

This release wraps up the initiative to ensure we capture all changes to Hedera entities. One of the oldest tickets in the repository going back to 2019 was completed, finally persisting the FreezeTransaction details to the database. There's a new option to store the raw TransactionRecord protobuf bytes that is set to off by default. The custom fee table was split into separate main and history tables for consistency with other data and improved querying efficiency.

An asynchronous database migration was added to efficiently update every account's crypto allowance amount after support for live allowance tracking was implemented in the last release. Furthermore, new crypto and fungible acceptance tests verify the live allowance tracking works correctly. Finally, we now rerun conditional migrations that would historically run only on initial startup. For migrations like balance initialization this means we automatically correct account and token relationship balances after ingesting the first balance file. For other migrations, it means they are triggered automatically based upon a specific record file version being ingested.

The REST API had a couple of noticeable changes. We now show only active allowances in the `/api/v1/accounts/{id}/allowances/crypto` and `/api/v1/accounts/{id}/allowances/tokens` REST APIs, providing consistency with how consensus nodes return this data. The `/api/v1/network/stake` API saw a change in how its stake value is calculated by changing it to stake rewarded plus not rewarded.

HIP-584 EVM Archive Node saw a number of HederaTokenService precompiles implemented including allowance, getApproved, isApprovedForAll, updateTokenExpiryInfo, updateTokenInfo, and updateTokenKeys. A large focus on testing resulted in increased integration and acceptance test coverage. The extra coverage resulted in a number of bugs being found and squashed, improving the reliability of `/api/v1/contracts/call`.

A lot of work went into the operations side of things as well. A good number of metrics and Grafana dashboards saw cleanup and improvements to aid in production monitoring. All chart dependencies saw version bumps and configuration adjustments to bring them up to date. Kubernetes 1.27 compatibility was confirmed as a deployment target while still ensuring backwards compatibility with prior Kubernetes versions. Compressed ZFS volume support now handles Kubernetes upgrades properly. Our Citus deployment saw an upgrade to Citus 12 with PostgreSQL 15. This release brings in the improvements we contributed upstream to Citus' createtimepartitions stored procedure so that it can support the bigint type that we use to store consensus timestamps. This allowed us to remove the pgpartman extension in favor of the native createtimepartitions. The pgcron extension was also removed in favor of a Java-based scheduled service running on the importer.

v0.86

There was a ton of progress towards our goal capturing all entity changes. This release adds the highly requested feature of tracking the remaining hbar and fungible token allowances and showing it via the amount field on the REST API. Importantly, this tracking only applies to new or updated allowances. Existing allowances will see their remaining allowance adjusted appropriately in the next release.

Traditionally, the mirror node has only stored the aggregated transfers from the transaction record. Now in addition we store the itemized transfers from the transaction body by default and embed it within the transaction table. For partial mirror nodes, we now create entities during balance initialization. This means even if a mirror node starts up now this migration can be rerun to create every account and contract with accurate balance information. In the next release, we'll extend this to automatically rerun this migration when processing the first account balance file.

Finally, we added an entitytransaction join table to start tracking all entities associated with a transaction. This will enable us provide a better transaction search experience and find all transactions associated with an entity. This functionality is disabled in this release as we iterate on it to make it performant.

Support for Ethereum transaction type 1 as specified in EIP-2930 is now

available. Previously only legacy and type 2 Ethereum transactions were supported. The new EIP-2930 transactions can be sent either directly to HAPI via an EthereumTransaction or via the JSON-RPC Relay.

HIP-584 EVM Archive node saw a large number of precompiles implemented. This includes all of the following:

- approve
- approveNFT
- associateToken
- associateTokens
- burnToken
- createFungibleToken
- createFungibleTokenWithCustomFees
- createNonFungibleToken
- createNonFungibleTokenWithCustomFees
- deleteToken
- freezeToken
- pauseToken
- setApprovalForAll
- transferFrom
- transferFromNFT
- transferNFT
- transferNFTs
- transferToken
- transferTokens
- unfreezeToken
- unpauseToken

v0.85

With the recent testnet reset that occurred on 2023-07-27, a new cloud storage bucket was created to store the stream files generated by the consensus nodes. This release will now use that updated bucket name by default when the network is set to testnet.

There were a few optimizations done specifically for the JSON-RPC Relay in this release that other users might also find useful. The `/api/v1/accounts/idOrAddress` REST API now supports an optional transactions query parameter that when set to false will omit the list of nested transactions. It defaults to true to match the previous behavior. If you're not using the transactions from this API please consider setting it to false to reduce the amount of data returned and provide an improved response time for your queries. Additionally, the `/api/v1/contracts/results` REST API was updated to include more fields to match the detailed results returned from `/api/v1/contracts/results/{id}`.

HIP-584 EVM Archive Node is adding functionality at a steady pace. This release adds support for the Ethereum Shanghai EVM version that adds a new PUSH0 opcode. Estimate gas calls that were failing for contract creates over 6 KB were fixed. Much of the focus was on implementing various precompiles including: approve allowance, burn, create, delete allowance, dissociate, grant KYC, revoke KYC, and wipe.

Making progress on capturing all state changes, we now keep a history of an account's daily entity stake. In the future, this information will be used to provide an accurate staked amount when looking up an account's historical information using `/api/v1/accounts/{id}?timestamp=`.

There were a lot of other bug fixes and small improvements. Please see the release notes below for the full list.

v0.84

This release contains support for HIP-729 contract nonce externalization. Consensus nodes now track and externalized contract nonce information in the record stream. A contract's nonce increases whenever it creates another contract. Mirror nodes persist this data and expose a contract's nonce on the `/api/v1/contracts` and `/api/v1/contracts/{id}` REST APIs.

HIP-584 EVM archive node continues to make progress. This release contains full support for contract state reads for both precompiled and non-precompile functions. Support was added for contract state modifications for non-precompile functions with the exception of lazy account creation.

Our goal of capturing all entity changes saw further refinements. Token information saw a history table be added to keep track of any changes to the token metadata over time. Also, we now use the real-time token balance in the token balances REST API.

v0.83

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JULY 7, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JUNE 29, 2023  
{% endhint %}
```

In this release we made the highly requested change to show the list of NFT transfers on the transactions list REST API. Originally, only the `/api/v1/transactions/{id}` showed the list of nfttransfers due to performance concerns with joining on another large table for such a heavily used and heavyweight API. To show this information while staying performant, we had to denormalize the NFT transfer information to nest it under the transaction table as a JSONB column. This avoids an extra join and allows us to return the given information with the existing query.

The mirror node is focused on tracking all possible changes to Hedera entities over time. To that end a NFT history table was created to capture all possible changes to a NFT over time. In addition to persisting this data, we're also exposing more of this historical information via the API. Now when the timestamp parameter is supplied on the `/api/v1/accounts/{id}` endpoint it will show the historical view of that account. Previously, the parameter would only be used for displaying the list of transactions at the given time. Expect additional improvements around historical entity information in the next release.

HIP-584 continues to make strides every release. This release focused on improving precompile support for `/api/v1/contracts/call`. There is now support for the CREATE2 opcode along with non-static contract state reads for precompile and non-precompile functions. Non-static contract modifications for non-precompile functions (excluding lazy account creation) was also worked on. Finally, acceptance test coverage was greatly increased and a number of bugs were addressed.

This release adds integration with the Stackgres Operator to provide a highly available Citus deployment. Stackgres is an established Kubernetes operator for PostgreSQL and their support for the Citus extension has made it easy to provide a production ready deployment without depending upon expensive, cloud-specific managed database services. This along with the ZFS volume compression we added in a previous release should greatly reduce the total cost of running a mirror node while providing horizontal scalability.

Upgrading

Expect upgrading to take about an hour due to the large NFT transfer migration in this release. As always, it's recommended upgrades be completed in a staging

environment first (e.g. a red/black deployment) to allow for deployment verification and reduce downtime before being opened up to customer traffic.

v0.82

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JUNE 22, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JUNE 8, 2023  
{% endhint %}
```

HIP-679 saw its initial work completed in this release to support a restructured bucket. The importer now supports the existing account ID-based bucket path along with a future node ID-based bucket path. It also adds a path type property that can automatically switch between the two so the transition between the two formats is transparent to mirror node operators. For now, the default path type will stay as account ID until node ID becomes a reality to reduce the S3 costs.

HIP-584 Mirror EVM Archive Node saw a large number of improvements to bring it closer to parity with consensus nodes. Stacked state and database accessors were integrated to allow for smart contracts to change state temporarily. An operation tracer was added to make it easier to debug smart contracts in an environment.

Finally, a topic message lookup table was added to optimize finding topic messages on distributed databases.

v0.81

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JUNE 1, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 24, 2023  
{% endhint %}
```

This release comes with a number of improvements to support the JSON-RPC Relay. We now support alias and EVM address lookups for the account.id parameter on the balances endpoint. We optimized the transaction nonce filtering in `/api/v1/contracts/{id}/results` by denormalizing the data. Finally, an issue with empty functionparameters in `/api/v1/contracts/results/{id}` response was addressed.

The other big item we worked on was support for volume level compression with Citus. We know that the type of time series data the mirror node stores would be highly compressible and we wanted to use that to our advantage to both reduce increasing storage costs and improve read/write performance. PostgreSQL supports a basic form of compression at the column level called TOAST, but it only takes effect for very large columns. Citus has compression when using their columnar storage access method, but we found it to be too slow for our needs. Since with Citus we knew we wouldn't be using a SaaS service we had more control over the database deployment, so we decided to experiment with Kubernetes volume compression. By creating custom Kubernetes node pools exclusively for Citus, we could install ZFS via the `zfs-localpv` and enable Zstandard compression on the database's underlying volume. The results were a 3.6x compression ratio with zero loss in performance. To put into perspective, that means the current mainnet database size of 17TB could be reduced to 4.7TB.

Other areas of improvement include improving documentation around disaster recovery efforts. This includes a runbook on restoring a mirror node from backup. There's also a runbook on how to perform local stream file verification.

Acceptance tests have been previously integrated into the automated deployment process but suffered from a long execution time mainly due to using Gradle to download dependencies at runtime. We containerized the acceptance tests so the dependencies are downloaded at build time reducing runtime by 3-4x.

v0.80

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MAY 17, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 11, 2023  
{% endhint %}
```

Work continues on HIP-584 with this release the first to support non-static contract state reads for non-compile functions. Please see the Swagger UI table for `/api/v1/contracts/call` for a breakdown of which functionality is supported in what release. More estimate gas functionality was copied from services code to make progress on estimation. A new stacked state frame functionality was added to be used in the future to support contract writes and cached reads.

The Spotless code formatting tool was used to format the entire codebase to be consistent. A git commit hook was added to ensure any new changes stays consistent and developers can focus on what matters.

Finally, there were a large number of bug fixes and performance improvements. See below for the full details.

v0.79

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MAY 9, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 2, 2023  
{% endhint %}
```

HIP-584 EVM archive node saw further progress this release with a focus on testing and establishing the foundation for estimate gas functionality in the next release. While consensus nodes undergo a modularization effort that will pay dividends down the road, the archive node needs functionality for estimate gas before that process could be completed. To make progress on HIP-584, the necessary EVM logic was temporarily copied from consensus nodes into the mirror node web3 module. A large focus was placed on increasing acceptance test coverage for contract call with precompiles.

Users writing dApps want to be able to monitor for token approval and transfer events. HAPI transactions like `CryptoTransfer`, `CryptoApproveAllowance`, `CryptoDeleteAllowance`, `TokenMint`, `TokenWipe`, and `TokenBurn` do not emit events that could be captured by monitoring tools like The Graph since they're executed outside the EVM. To address, the mirror node now generates synthetic contract log events for these non-EVM HAPI transactions.

A new subscription API was designed for HIP-668 GraphQL API. Once it's implemented in a future release, the new contract log subscription will stream contract events to clients via a WebSocket connection.

For our Citus database transition, PostgreSQL 15 compatibility was verified and made the default for this v2 schema. The lookup of historical balance information via `/api/v1/balances?timestamp=` was optimized for sharded databases so it stays performant. Performance testing showed a decrease in shard count

could greatly improve performance so we lowered the number of shards from 32 to 16. This testing also allowed us to provide an initial recommended resource configuration for the Citus deployment.

There was a large focus on test improvements in this release. In addition to the aforementioned HIP-584 test coverage, we also optimized the acceptance tests to reduce the overall test duration in Kubernetes by half without reducing coverage. The acceptance test logs were cleaned up to reduce unnecessary log statements and standardize its output. The hbar balance used by the tests now is logged at the end of test execution. Acceptance tests for hollow account creation were added. We now generate multi-platform snapshot images from the main branch for testing with local node. Testkube configuration was enhanced to make it more configurable. Finally, all Java test compiler warnings were fixed and will now fail the build if any future warnings occur.

Known Issues

There is a bug introduced by #5776 that causes the importer to fail on startup. It's recommended to hold off on upgrading to v0.79.0 until we can address this in a v0.79.1. Alternatively, it can be worked around by disabling the faulty migration by setting `hedera.mirror.importer.migration.syntheticTokenAllowanceOwnerMigration.enabled=false`.

v0.78

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: APRIL 21, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: APRIL 13, 2023  
{% endhint %}
```

HIP-584 Mirror Evm Archive Node now has token precompile support. This was the last major piece of functionality needed for the `/api/v1/contracts/call` to be considered ethcall equivalent. The new API was added to the REST API's OpenAPI documentation so that it appears on our Swagger UI. A number of performance optimizations were worked on to make it scalable as well as various test improvements to verify its correctness. Various bugs were addressed including the proper handling of reverts. In the next few releases, we plan to fine tune contract call and implement contract gas estimation.

A large focus was put on performance and resiliency this release. On the performance front, we've optimized the list schedules REST API to be scalable on Citus. Performance tests can now trigger automatically via TestKube once the helm tests complete. Those same k6 performance tests were enhanced to automatically pick appropriate configuration values specific to the environment. The transaction hash table was partitioned and the ingest process made to insert hashes in parallel. This change dramatically speeds up the time to insert the optional transaction hashes. Similarly, an option was added to control which transaction types should cause a hash insertion.

On the resiliency front, the importer component was analyzed for any code paths that may cause record file processing to halt due to bad input from consensus nodes. Any such code was made to handle the error, log/notify, and move on to the next transaction. This change makes the mirror node ingestion more resilient and moves toward preferring availability over correctness. Partial mirror nodes that might become stuck due to having an incomplete address book can now continue to ingest with a new consensusMode property and logic. Partial mirror nodes will now also be able to have a corrected account and token balance even if the entity was missing a deleted flag. Finally, we were able to complete a longstanding refactoring effort to move all transaction specific logic to individual transaction handlers and fixed a number of bugs in the process.

There were a few important bugs fixed in this release that are worth noting. A fix was put in place to correct inaccurate fungible token total supply. Additionally, NFTs for deleted tokens no longer appear as active in the REST API.

v0.77

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: APRIL 4, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MARCH 29, 2023  
{% endhint %}
```

This release fixes the tracking of NFT balances. Historically, these came from the balance file sent by the consensus nodes every 15 minutes. When we started tracking the live fungible token balances and moved away from using this balance file we unfortunately broke the NFT balance calculation. We not only fixed the issue but went ahead and took the time to track the up-to-date NFT balance as well.

The `/api/v1/contracts/{id}/state` REST API shows the current state of a contract's slot values. Users requested the ability to query for the key/value pairs for their contract at an arbitrary point in the past. To address, we now expose a timestamp query parameter that will get the historical contract state. This allows the JSON-RPC relay to offer a proper `ethgetStorageAt` with support for historical blocks.

HIP-584 continues to make progress. Quite a few bugs were squashed including handling reverts and populating the revert reason and raw data. Performance tests were added to k6 to load test contract calls with token precompiles.

v0.76

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MARCH 23, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MARCH 13, 2023  
{% endhint %}
```

The new `/api/v1/contracts/call` REST API as specified in HIP-584 is finally ready for initial production use. This release adds support for rate limiting the API with an initial value of 100 requests per second per instance. Tags were added to the gas per second metric to indicate if the request was a call, an estimate, or resulted in an error for increased observability. Various bug fixes were also addressed.

HIP-668 GraphQL API was added to our deployment with the addition of a new helm chart for this API. This will allow for initial use of the API in all environments with the understanding that it's still very much alpha and subject to change.

We made a lot of headway on our Citus integration. Citus was upgraded to 11.2 which showed a nice 15-20% performance boost for a number of query patterns. We optimized the contract results APIs performance by distributing based upon contract ID instead of payer account. Search for a transaction by its hash on Citus was improved by adding the distribution column to the query and limiting the results to a timestamp range. The search for an account by its ID also saw

improvements on Citus.

Breaking Changes

The Helm charts contain some breaking changes to be aware of. The hedera-mirror wrapper chart enables the new hedera-mirror-graphql sub-chart by default. The GraphQL deployment requires a new mirrorgraphql database user to exist for it to successfully start up. You can create the user by logging into the database as owner and executing the following SQL query:

```
create user mirrorgraphql with login password 'password' in role readonly;
```

If you're using the embedded PostgreSQL sub-chart (which we don't recommend for production use), then you'll have to delete its StatefulSet before upgrading due to a major bump in its chart version.

The hedera-mirror-common chart had all of its components upgraded to new major versions that include breaking changes. If you're using this chart, run the following commands before upgrading:

```
kubectl delete daemonset mirror-prometheus-node-exporter
kubectl delete daemonset mirror-traefik
kubectl delete statefulset mirror-loki
kubectl delete ingressroute mirror-traefik-dashboard
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/
monitoring.coreos.comalertmanagerconfigs.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/monitoring.coreos.comalertmanagers.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/monitoring.coreos.compodmonitors.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/monitoring.coreos.comprobes.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/monitoring.coreos.comprometheuses.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/
monitoring.coreos.comprometheusrules.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/
monitoring.coreos.comservicemonitors.yaml
kubectl apply --server-side --force-conflicts=true -f
https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/
v0.63.0/example/prometheus-operator-crd/monitoring.coreos.comthanosrulers.yaml
```

v0.75

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: MARCH 2, 2023
{% endhint %}
```

Work continues on HIP-584 to get it closer to production ready for simple contract calls. Caching logic was added to the repository layer to optimize its

capability along with performance tests to verify those improvements. A metric was added to track the gas per second being used along with various other bug fixes.

The monitor API and dashboard used internally for observing our production systems was containerized. Additionally, it was integrated into the Helm chart and invoked as part of the Helm tests to ensure the deployment is verified.

Finally, there were a number of query optimizations as part of our Citus effort.

v0.74

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: FEBRUARY 18, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: FEBRUARY 14, 2023  
{% endhint %}
```

This release switches the testnet bucket to the new one created for the testnet reset that occurred on January 26, 2023. It also updates the address book to reflect the additional nodes added to testnet since the last reset. If you're running a testnet mirror node, please see the reset instructions for help getting your node updated.

In HIP-668, we propose adding a new mirror node GraphQL API and would greatly appreciate your feedback. In this release, a new GraphQL module with a simple account lookup query was added to provide the basis for future work on this HIP. In the next release, we will add the automated deployment of this module to all environments. It is considered an alpha API subject to breaking changes at any time, so it's not recommended to depend upon for production use. This has been made explicit by using /graphql/alpha in its URL.

Finally, a number of query optimizations were implemented for Citus while ensuring it doesn't cause regressions with the existing database. This will continue to be the focus of the next few releases.

v0.73

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: FEBRUARY 10, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: FEBRUARY 3, 2022  
{% endhint %}
```

This release is the first one with support for HIP-584 EVM Archive Node. HIP-584 allows mirror nodes to act as a read only EVM for free execution of smart contracts. This new feature is considered alpha with much of the work still to be implemented like support for precompiled contracts, gas estimation, etc. This functionality requires the mirror node to be configured to ingest the optional traceability sidecar files and it requires a network where those files are generated. Currently only previewnet has contract traceability enabled.

The testnet bucket name has been updated to the new bucket name after its recent quarterly reset. Likewise the bootstrap testnet address book was updated to reflect the additional testnet nodes that have been added since the previous reset. Mirror node operators running a testnet node should either manually populate the new bucket name or update to this release.

The remaining work targeted significant testing improvements and bug fixes. Our performance tests were expanded to all endpoints to catch issues earlier in the

lifecycle. Additional acceptance test coverage was added along with a number of fixes. CI stability has greatly improved with a focus on fixing flaky tests. Code smells as reported by Sonar were reduced to only a handful and in the next release reduced all the way down to zero. Finally, we merged work that enables nightly performance testing in our integration and mainnet staging environments via TestKube.

v0.72

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JANUARY 25, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JANUARY 18, 2022  
{% endhint %}
```

This release is a smaller release as most of the team took time off for the holidays. Still, we managed to implement HIP-583 Expand alias support in CryptoCreate & CryptoTransfer Transactions. We now allow hollow accounts to be later finalized into a contract when it is fully created.

We also worked on adding support for Testkube. Testkube allows us to automate our testing in Kubernetes environments by triggering tests based upon various conditions. Specifically, it will be used to run nightly performance regression tests against a mainnet staging environment to ensure our API performance doesn't regress. We'll continue to expand on this automated testing in future releases.

There were also a number of bug fixes in this release, mainly focused on fixing our release process after the switch from Maven to Gradle in the last release.

v0.71

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JANUARY 19, 2023  
{% endhint %}
```

As of this release, all account and token balances in the REST API will reflect their real time balance information. Historically, the mirror node has relied upon the balance file uploaded by consensus nodes every 15 minutes for its balance information. We've been working towards this milestone for many releases gradually rolling out real-time balance tracking to more entities and more APIs. This release completes this migration with the addition of real time token balances to both the accounts and the balances REST APIs.

The mirror node now implements support for HIP-583 alias on CryptoCreate transactions. With this, clients can directly set an alias during account creation instead of relying upon the implicit auto-account creation during transfers. The mirror node respects this explicit alias along with the new explicit EVM address in both CryptoCreate or the TransactionRecord. This avoids the brittle EVM address calculation on the mirror node that has caused us some trouble in the past.

This release completes the migration from Maven to Gradle for our build process. A lot of work has been put into the new build to improve its performance and stability both locally and in continuous integration (CI). GitHub Actions workflows have been consolidated from one workflow per module to a single Gradle build workflow with a matrix strategy running them in parallel for each module and database schema. This greatly simplifies the workflow configuration making it easier to maintain and debug.

We continue to make progress on our Citus exploration. The v2 schema for Citus now does timestamp based partitioning of data and automates this process via pg\

cron. A Citus specific environment was created and we're currently conducting performance tests against it at scale to verify it meets our requirements.

This release adds automation to keep our GCP Marketplace application up to date with each release. While not fully automatic due to the manual nature of Marketplace version submission, now any new production tag will trigger the generation and verification of the marketplace images.

This was a big release and there were a lot of other various improvements and fixes. See the full release note below.

v0.70

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: DECEMBER 29, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: DECEMBER 14, 2022
{% endhint %}
```

As part of HIP-406, the mirror node is adding a new account staking rewards REST API. This API will show the staking rewards paid to an account over time. The mirror node now also shows staking reward transfers in the transaction REST APIs (e.g. /api/v1/transactions, /api/v1/transactions/{id}, and the list of transactions in /api/v1/accounts/{id}). This can be useful to show which transaction involving your account after the staking period ended triggered the lazy reward payout.

GET /api/v1/accounts/{id}/rewards

```
{
  "rewards": [{
    "accountid": "0.0.1000",
    "amount": 10,
    "timestamp": "123456789.000000001"
  }],
  "links": {
    "next": null
  }
}
```

The REST API saw further improvements outside of staking. The accounts REST APIs now show a calculated expiration timestamp to mirror the HAPI CryptoGetInfo query. Previously expiration timestamp only shows up if explicitly sent via a transaction that supports it (mainly update transactions). Now if it's null we calculate it as createdtimestamp.seconds + autorenewperiod. Every contract results endpoint was updated to include an address field for the EVM address of the created contract.

This release makes progress on being able to execute contract calls on the mirror node as outlined in HIP-584. A lot of the groundwork is being laid that will be further refined in upcoming releases.\

v0.69

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: DECEMBER 5, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: NOVEMBER 29, 2022
```

```
{% endhint %}
```

As noted in previous releases, HIP-367 is deprecating the token relationship information returned from HAPI queries. In this release, its mirror node replacement is now feature complete. We now track and show the current fungible token balance in the token relationships API instead of relying upon the 15 minute balance export from consensus nodes. In a future release, the accounts and balances REST APIs will be updated to show the current fungible token balance.

The importer component now supports a local file stream provider. This allows it to read stream files from a local directory instead of just the S3-compatible providers it supported previously. This mode is useful for debugging stream files received out of band or for reducing complexity and latency in a local node setup. To try it out, set `hedera.mirror.importer.downloader.cloudProvider=LOCAL` and populate the `hedera.mirror.importer.dataPath/streams` folder with the same file structure as the cloud buckets.

We now show a contract's CREATE2 EVM address in the contract logs REST APIs. Previously, we would convert the Hedera shard.realm.num to a 20-byte EVM address but this did not always reflect the true EVM address of the contract. Using the CREATE2 form of the EVM address provides increased Ethereum compatibility.

We continue to make progress on converting our build process to Gradle. This release adds a Golang Gradle plugin to download the Go SDK and use it to build and test the Rosetta module.

v0.68

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: NOVEMBER 18, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 18, 2022  
{% endhint %}
```

Besides the usual round of bug fixes, this release focuses on some internal enhancements to lay the groundwork for some upcoming features. We now track and persist the current fungible token balance in the database. This information is not yet exposed on any API but will be rolled out to the token relationships, accounts and balances REST APIs in the near future.

We're continuing our work towards CitusDB as a possible database replacement in this release by adding distribution columns and fixing our v2 schema tests.

Finally, we implemented initial Gradle support to improve build times and provide a better developer experience. Initial testing shows build and test times reduced from 8 minutes overall down to 2 minutes. The Gradle and Maven build scripts will be maintained concurrently for a few releases until we can ensure the Gradle build reaches feature parity with Maven.

v0.67

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: NOVEMBER 10, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 10, 2022  
{% endhint %}
```

HIP-367 deprecated the list of token balances for an account returned via HAPI.

account ID to use its node ID instead.

On the testing front, we enhanced various test and monitoring tools to add support for new APIs. We also added an acceptance test startup probe to delay the start of the tests until the network as a whole was healthy. This avoids the mirror node acceptance tests reporting a false positive when a long migration or startup process on the consensus or mirror nodes causes a delay.

v0.66

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: OCTOBER 24, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: OCTOBER 17, 2022  
{% endhint %}
```

Continuing our goal of having up to date balances everywhere, this release now shows the current hbar balance on the balances REST API. If you provide a timestamp parameter, it will fallback to the previous behavior and use the 15 minute balance file. This allows us to continue to provide a historical view of your balance over time, while also showing the latest balance if no specific time range is requested. Live fungible token balance is actively being worked on for an upcoming release.

Another big feature this release is support for cloud storage failover. The importer can now be configured with multiple S3 download sources and will iterate over each until one is successful. This makes the mirror node more decentralized and provides more resiliency in the face of cloud failure. The existing `hedera.mirror.importer.downloader` properties used to configure the `cloudProvider`, `accessKey`, `secretKey`, etc. will continue to be supported and inserted as the first entry in the sources list, but it's recommended to migrate your configuration to the newer format. Also in the downloader, we increased the downloader batch size to 100 to improve historical synchronization speed. A `hedera.mirror.importer.downloader.sources.connectionTimeout` property was added to avoid occasional connection errors.

```
hedera:  
  mirror:  
    importer:  
      downloader:  
        sources:  
          - backoff: 30s  
            connectionTimeout: 10s  
            credentials:  
              accessKey: <redacted>  
              secretKey: <redacted>  
            maxConcurrency: 50  
            projectId: myapp  
            region: us-east-2  
            type: GCP  
            uri: https://storage.googleapis.com  
          - credentials:  
              accessKey: <redacted>  
              secretKey: <redacted>  
            type: S3
```

HIP-573 gives token creators the option to exempt all of their token's fee collectors from a custom fee. This mirror node release adds support to persist this new `allcollectorsareexempt` HAPI field and expose it via the `/api/v1/tokens/{id}` REST API.

An important characteristic of a mirror node is that anyone can run one and store only the data they care about. The mirror node has supported such capability for a few years now but the configuration syntax was a bit tricky to get correct. To address this shortcoming, we add some examples to the configuration documentation to clarify things. This entity filtering was historically limited to just create, update and delete operations on entities. We've now expanded this filtering to include payer account IDs and accounts or tokens involved in transfers.

Breaking Changes

As part of the S3 failover work, we made a number of changes to existing properties to streamline things and only support one property for all stream types. The `hedera.mirror.importer.downloader.(balance|event|record).batchSize` properties were removed in favor of a single, generic `hedera.mirror.importer.downloader.batchSize`. Likewise, the `hedera.mirror.importer.downloader.(balance|event|record).threads` properties were removed in favor of `hedera.mirror.importer.downloader.threads`. The `hedera.mirror.importer.downloader.(balance|event|record).prefix` properties were removed in favor of hardcoded configuration since there's never been a need to adjust these. If you're using any of these properties, please adjust your config accordingly.

If you're writing stream files to disk after downloading by enabling the `writeFiles` or `writeSignatures` properties, there is one other breaking change to be aware of. As part of our migration away from node account IDs, we changed the paths on disk to use the node ID as well. If you'd like to avoid having two directories for the same node please rename your local directories manually. For example, change `${dataDir}/recordstreams/record0.0.3` to `${dataDir}/recordstreams/record0`.

v0.65

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: OCTOBER 11, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: OCTOBER 6, 2022
{% endhint %}
```

The mirror node now calculates consensus using the staking weight of all the nodes as outlined in HIP-406. If staking is not yet activated, it falls back to the previous behavior of counting each node as 1/N weight where N is the number of consensus nodes running on the network. The `/api/v1/accounts` and `/api/v1/accounts/{id}` REST APIs now expose a `pendingreward` field that provides an estimate of the staking reward payout in tinybars as of the last staking period. The `/api/v1/network/supply` REST API updated its configured list of unreleased supply accounts to accurately reflect the separation of accounts done for staking purposes by Hedera.

This release implements the contract actions REST API detailed in HIP-513. An example of an actions payload is shown below. We added `transactionhash`, `transactionindex`, `blockhash` and `blocknumber` fields to the contract logs REST APIs. This work was done to optimize the performance for the `ethgetLogs` JSON-RPC method used by the relay. Also for the relay, we now support lookup of non-Ethereum contract results by its 32 byte transaction hash.

GET `/api/v1/contracts/results/0.0.5001-1676540001-234390005/actions`

```
{
  "actions": [{
```


again. It also saw performance improvements including a delay property to throttle its speed and added job status persistence so it doesn't restart from the beginning every time. A new remediationStrategy property provides a mechanism to continue after failure to aid in debugging multiple reconciliation errors.

v0.63

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: SEPTEMBER 7 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: SEPTEMBER 2, 2022  
{% endhint %}
```

This release adds a highly requested feature: the mirror node now tracks the current account balance. Previously, the mirror node would store balance information whose source was a balance file that consensus nodes generate and upload every 15 minutes. As a result, balance information was always behind by up to 15 minutes for active accounts. We were able to figure out a way to track this information at scale with SQL in this release. The next release will actually expose this up to date account balance information on both `/api/v1/accounts` or `/api/v1/accounts/{id}`. In future releases, will look at adding live balances to `/api/v1/balances` when no timestamp parameter is provided and track up to date token balances.

Work continues on HIP-513 Contract Traceability, with this release adding a few important items. Consensus nodes will, when first activating the sidecar mechanism, send migration records that includes all smart contract runtime bytecode and current storage values. The mirror node now supports receiving these special migration sidecars and updating its database with the migrated data. This paves the way for the mirror node to have the necessary information to execute smart contracts without modifying state in a future release. Also in this release we now show the contract initcode that was used to unsuccessfully create a smart contract in a new `failedinitcode` field in the contract result REST API.

The network supply REST API saw an update to adjust the unreleased supply accounts used to calculate the unreleased supply. This change was necessary as Hedera adjusts the treasury accounts for use with staking.

v0.62

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: AUGUST 29, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 22, 2022  
{% endhint %}
```

Mirror Node 0.62 saw HIP-406 staking related improvements to its REST API and partial support for HIP-513 contract traceability.

The `/api/v1/network/nodes` will now use the address book stake as a fallback when it has not seen any `NodeStakeUpdate` transactions on the network. This release also contains a new network stake REST API `/api/v1/network/stake` to show aggregate stake information common to all nodes:

```
{  
  "maxstakingrewardrateperhbar": 17808,  
  "noderewardfee": 0.0,
```

```

    "staketotal": 35000000000000000,
    "stakingperiod": {
      "from": "1658774045.000000000",
      "to": "1658860445.000000000"
    },
    "stakingperiodduration": 1440,
    "stakingperiodsstored": 365,
    "stakingrewardfeefraction": 1.0,
    "stakingrewardrate": 100000000000,
    "stakingstartthreshold": 25000000000000000
  }
}

```

HIP-513 Smart Contract Traceability adds support for an optional sidecar to contain contract traceability information. In this release, the mirror node supports downloading and persisting contract state changes, contract initcode, contract runtime bytecode, and contract actions (AKA traces). The `/api/v1/contracts/{id}` REST API now shows the runtime bytecode for newly created contracts. The next release will support a sidecar migration that will populate contract state changes and bytecode for all existing contracts.

HIP-435 Record Stream V6 required changes to the state proof REST API in order to not break when V6 was enabled. With this release, the API was updated to support record files in the new v6 format.

The Rosetta API saw a few minor fixes and improvements. It now uses the Hedera network alias everywhere in the Rosetta server. It also fixes the issue that Rosetta did not support alias as the from address for crypto transfers. Additionally, the Rosetta subnetworkidentifier was disabled since it was not needed.

There were a surprising number of technical debt improvements this release. The REST API and monitor API were both converted from CommonJS to ES6 modules, allowing us to finally upgrade some of our dependencies to the latest version. The REST API spec tests were organization into folders by endpoint and changed to use a single database container for the entire suite. On the importer, mutable contract information was merged into the entity table. The RecordItem constructor was removed everywhere in favor its builder method. Finally, we added parser performance tests to be able to generate large record files and stress test record file ingestion.

Breaking Changes

In a recent release, we added the `staketotal` field to the `/api/v1/network/nodes` API to show the aggregate stake of the network. With the addition of the new `/api/v1/network/stake` API, we now have a separate API to return aggregate staking information associated with the network. As such, we made the decision in this release to remove the `staketotal` field from the response of the `/api/v1/network/nodes` API to stay consistent. If you're using this field, please update your code to use the `staketotal` field in the `/api/v1/network/stake` API.

v0.61

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: AUGUST 2, 2022
{% endhint %}

```

```

{% hint style="success" %}
TESTNET UPDATE COMPLETED: JULY 27, 2022
{% endhint %}

```

This release adds initial support for HIP-513 Smart Contract Traceability Extension. Contract traceability information is now available inside an optional sidecar file uploaded separately to cloud storage. Mirror node operators can

choose whether to download this extra information by configuring the `hedera.mirror.importer.parser.record.sidecar` properties on the importer. By default, sidecar files will not be downloaded. Enabling it will permit contract state, actions, and bytecode data to be persisted by the mirror node. HIP-513 support is incomplete in this release and the next release will enable full persistence of all sidecar types.

The transactions REST API now supports multiple `transactiontype` query filters to simplify searches across types.

The version of the mirror node in GCP Marketplace was updated to v0.60.0. This required migration to the new GCP Producer Portal which should help streamline future version updates.

The monitor components saw an option added to retrieve the address book on startup. This avoids having to configure the list of nodes to monitor manually in pre-production environments and ensure the list of nodes is up to date. The monitor now uses OpenAPI generated models to dog food our OpenAPI schema. We also added an option to the monitor to set the max memo length property for published transactions.

v0.60

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JULY 18, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JULY 14, 2022  
{% endhint %}
```

The two big features of this release are support for a data retention period and HIP-351 pseudorandom number generation.

On public networks, mirror nodes can generate tens of gigabytes worth of data every day and this rate is only projected to increase. Mirror nodes now support an optional data retention period that is disabled by default. When enabled, the retention job purges historical data beyond a configured time period. By reducing the overall amount of data in the database it will reduce operational costs and improve read/write performance. Only insert-only data associated with balance or transaction data is deleted. Cumulative entity information like accounts, contracts, etc. are not deleted.

HIP-351 adds a pseudorandom number generator transaction. The mirror node now persists this `PrngTransaction` type including the pseudorandom number or the bytes it generates. A future release will expose this information on the REST API.

There were various other improvements in this release. Block numbers are now migrated to be consistent with other mirror nodes regardless of their configured start date when it receives the first v6 record file with the canonical block number from consensus nodes. We added the reward rate at the start of the staking period to the nodes REST API. Rosetta now shows fee crypto transfers operation type as `FEE`. Rosetta also shows account aliases as account addresses in Rosetta DATA API response.

v0.59

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JUNE 29, 2022  
{% endhint %}
```

The previous release saw support for the persistence of HIP-406 staking-related data. Staking persistence saw further fine-tuning in this release to adapt to

changes in the NodeStakeUpdateTransaction protobuf. The declinereward, stakedaccountid, stakednodeid fields were added to /api/v1/accounts and /api/v1/accounts/{id} to show account-level staking properties. We also added staking related fields to the existing /api/v1/network/nodes REST API (see example below).

GET /api/v1/network/nodes

```
{
  "nodes": [
    {
      "description": "address book 1",
      "fileid": "0.0.102",
      "maxstake": 50000,
      "memo": "0.0.4",
      "minstake": 1000,
      "nodeaccountid": "0.0.4",
      "nodecerthash": "0x01d...",
      "nodeid": 1,
      "publickey": "0x4a...",
      "serviceendpoints": [],
      "stake": 20000,
      "stakenotrewarded": 19900,
      "stakerewarded": 100,
      "staketotal": 100000,
      "stakingperiod": {
        "from": "1655164800.000000000",
        "to": "1655251200.000000000"
      },
      "timestamp": {
        "from": "16552512001.000000000",
        "to": null
      }
    }
  ],
  "links": {
    "next": null
  }
}
```

Support for the new record file v6 format as defined in HIP-435 was added in this release. Record file v6 adds block number as well as support for the new sidecar record files that carry detailed contract traceability information that mirror nodes can optionally choose to download. The record and signature files are now in a more maintainable protobuf format that should make them easier to enhance with new fields in the future without requiring breaking changes. Also, the v6 record files will now be compressed which should translate into reduced network and storage costs while potentially improving performance. Once v6 is enabled in a future hedera-service's release, mirror node operators will be required to update to a version that supports the new v6 format to avoid downtime.

Rosetta saw a number of improvements this release to better align it with the Rosetta specification. A configurable valid duration seconds option was added to the transaction construction API to support customization of this value. Support for a consistent block number regardless of startDate was added in Rosetta now that Hedera has a consistent block as defined in HIP-415. A 0x prefix was added to alias addresses returned via the API to denote that the data is hex-encoded.

v0.58

```
{% hint style="success" %}
```

MAINNET UPDATE COMPLETED: JUNE 22, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE COMPLETED: JUNE 16, 2022
{% endhint %}

This release contains support for HIP-406 Staking, HIP-410 Wrapped Ethereum Transaction, and HIP-482 JSON-RPC Relay as well as a long overdue upgrade to Java 17.

HIP-406 Staking is coming and the mirror node is getting ready for it. This release we added persistence support to store staking information. In the next release, we'll expose this information via our APIs.

HIP-410 and HIP-482 are both intended to improve the onramp for existing Ethereum developers. Towards that end, we added pagination support to both of the contract logs REST APIs. You can now page through logs via a combination of a consensus timestamp and log index parameters. The new blocks REST APIs also saw new gasused and logsbloom fields added that show the aggregated values for all transactions within the block. Finally, we added a new network fee schedule REST API. Currently, it only exposes the gas price for ContractCall, ContractCreate, and EthereumTransaction types in tinybars.

GET /api/v1/network/fees

```
{
  "fees": [
    {
      "gas": 35561,
      "transactiontype": "ContractCall"
    },
    {
      "gas": 481934,
      "transactiontype": "ContractCreate"
    },
    {
      "gas": 35561,
      "transactiontype": "EthereumTransaction"
    }
  ],
  "timestamp": "1633392000.387357562"
}
```

Since mirror node's inception in 2019, it has used Java 11 to build and run due to it being the most recent LTS release. After Java 17 LTS was released in September 2021 we knew we wanted to upgrade. With this release we upgraded to 17 after validating that the mirror node was still functional and performant. If you're using our official container images, they are also on Java 17 so there will be no migration necessary besides updating the image. If you're running outside of a container, you'll need to either upgrade your JRE to 17 or rebuild the jar from source with -Djava.version=11.

v0.57

{% hint style="success" %}
MAINNET UPDATE COMPLETED: MAY 25, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE COMPLETED: MAY 25, 2022
{% endhint %}

This release is focused on adding the necessary data and APIs needed for the JSON-RPC Relay defined in HIP-482. The JSON-RPC Relay implements the Ethereum JSON-RPC standard and relays HIP-410 Ethereum transactions to consensus nodes. Since the concept of a block is crucial for JSON-RPC APIs, this release also contains the implementation of HIP-415 Introduction of Blocks.

The mirror node now exposes the concept of blocks as introduced in HIP-415. We now calculate and store the cumulative gas used and the contract log bloom filter for the block as a whole. This HIP defines three new REST APIs and this release includes all three: a list blocks REST API, a get blocks REST API, and a list contract results REST API. The new `/api/v1/blocks` API supports the usual limit and order query parameters along with timestamp and block.number to support equality and range operators for consensus timestamps and block numbers, respectively. The `/api/v1/blocks/{hashOrNumber}` is identical to the list blocks but only returns a single block by either its block hash or its block number. Finally, a `/api/v1/contracts/results` REST API was added that is identical to the existing `/api/v1/contracts/{id}/results` but able to search across contracts.

GET `/api/v1/blocks`

```
{
  "blocks": [{
    "count": 4,
    "gaslimit": 150000000,
    "gasused": 50000000,
    "hapiversion": "0.24.0",
    "hash":
"0xa4ef824cd63a325586bfe1a66396424cd33499f895db2ce2292996e2fc5667a69d83a48f3883f
2acab0edfb6bfeb23c4",
    "logsbloom": "0x549358c4c2e573e02410ef7b5a5ffa5f36dd7398",
    "name": "2022-04-07T165923.159846673Z.rcd",
    "number": 19533336,
    "previoushash":
"0x4fbcefec4d07c60364ac42286d5dd989bc09c57acc7370b46fa8860de4b8721e63a5ed46addf1
564e4f8cd7b956a5afa",
    "size": 8489,
    "timestamp": {
      "from": "1649350763.159846673",
      "to": "1649350763.382130000"
    }
  }],
  "links": {
    "next": null
  }
}
```

GET `/api/v1/blocks/{hashOrNumber}`

```
{
  "count": 4,
  "gaslimit": 150000000,
  "gasused": 50000000,
  "hapiversion": "0.24.0",
  "hash":
"0xa4ef824cd63a325586bfe1a66396424cd33499f895db2ce2292996e2fc5667a69d83a48f3883f
2acab0edfb6bfeb23c4",
  "logsbloom": "0x549358c4c2e573e02410ef7b5a5ffa5f36dd7398",
  "name": "2022-04-07T165923.159846673Z.rcd",
  "number": 19533336,
  "previoushash":
```



```

"0x4fbcefec4d07c60364ac42286d5dd989bc09c57acc7370b46fa8860de4b8721e63a5ed46addf1
564e4f8cd7b956a5afa",
  "size": 8489,
  "timestamp": {
    "from": "1649350763.159846673"
    "to": "1649350763.382130000"
  }
}

```

A number of changes were made in support of HIP-410 Ethereum Transactions. The `/api/v1/accounts/{idOrAlias}` REST API was updated to accept an EVM address as a path parameter in lieu of an ID or alias. An `ethereumnonce` and `evmaddress` was added to the response of `/api/v1/accounts/{idOrAliasOrAddress}` and `/api/v1/accounts`. The existing `/api/v1/contracts/results/{transactionId}` was updated to accept the 32 byte Ethereum transaction hash as a path parameter in addition to the transaction ID that it supports now. Its response, as well as the similar `/api/v1/contracts/{idOrAddress}/results/{timestamp}`, was updated to add the following new Ethereum transaction fields:

```

{
  "accesslist": "0xabcd...",
  "blockgasused": 564684,
  "chainid": "0x0127",
  "gasprice": "0xabcd...",
  "maxfeepergas": "0xabcd...",
  "maxpriorityfeepergas": "0xabcd...",
  "nonce": 1,
  "r": "0x84f0...",
  "s": "0x5e03...",
  "transactionindex": 1,
  "type": 2,
  "v": 0
}

```

Note: Existing fields omitted for brevity.

A new exchange rate REST API `/api/v1/network/exchangerate` was added that returns the exchange rate network file stored in `0.0.112`. It supports a `timestamp` parameter to retrieve the exchange rate at a certain time in the past.

```

{
  "currentrate": {
    "centequivalent": 596987
    "expirationtime": 1649689200
    "hbarequivalent": 30000
  },
  "nextdate": {
    "centequivalent": 594920
    "expirationtime": 1649692800
    "hbarequivalent": 30000
  },
  "timestamp": "1649689200.123456789"
}

```

A new `/api/v1/contracts/results/logs` API was added with the same query parameters and response as `/api/v1/contracts/{address}/results/logs` but with the ability to search across contracts. It does not support `address` as a query parameter as it's expected users use the existing API if they need logs for a specific address. The same rules around not exceeding `maxTimestampRange` still

applies and allows it to stay performant. Pagination is possible using a combination of the timestamp and index query parameters.

Finally, this release completes our implementation of HIP-423 Long Term Scheduled Transactions. Two new fields `waitforexpiry` and `expirationtime` were added to `/api/v1/schedules` and `/api/v1/schedules/{id}`

v0.56

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MAY 18, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 17, 2022  
{% endhint %}
```

This is a big release with support for six different Hedera Improvement Proposals. Most of these changes are on the ingest side of things and future releases will work on adding them to our APIs.

HIP-16 will enable contract expiry and brings a new auto renew account field that contains the account responsible for any renewal fees associated with the contract. The contract REST APIs now show this `autorenewaccount` field along with a new `permanentremoval` flag that is set to true when the system expires a contract.

We missed a requirement in our implementation of HIP-329 CREATE2 opcode to allow a HAPI client to do a native transfer of assets to a contract known only by its CREATE2 address. We fixed this logic gap and now support EVM addresses in a `CryptoTransferTransaction`.

HIP-336 saw further polish to our allowance support. Support for an optional spender ID parameter on our `/api/v1/accounts/{id}/nfts?spender.id={id}` REST API is now included.

HIP-410 brings with it the ability to wrap an Ethereum native transaction and submit it to Hedera. The mirror node can now parse this new `EthereumTransaction` along with the results from its execution. Any contracts created or results and logs generated by its execution will automatically show up on the relevant contract REST APIs. Support for specifying the contract initcode directly on a contract create was also added. To support Externally Owned Accounts (EOA) being able to submit Ethereum transactions, we now calculate and store the EVM address of the account's ECDSA secp256k1 alias. Finally, we added support for repeated topics in contract logs REST API to bring it more inline with the `ethgetLogs` JSON-RPC method. If you supply multiple different topic parameters (e.g. `topic0` and `topic1`) it is considered an AND operation as before, but if pass repeated parameters like `topic0=0x01&topic0=0x02&topic1=0x03` it means "(topic 0 is 0x01 or 0x02) and (topic 1 is 0x03)".

HIP-415 is defining a block as the number of records files since stream start (AKA genesis). Since only mirror nodes have a full history, they will be used to provide consensus nodes the current block number to update their state. Since partial mirror nodes with an effective start date after stream start won't have all record files, they may contain an inconsistent value for their block number in contrast to other mirror nodes with all data. This release attempts to correct that with a migration to bring them inline with full mirror nodes so everyone has a consistent block number value.

HIP-423 Long term scheduled transactions enhances the existing scheduled transactions to allow time-based scheduling of transactions. This release adds ingest support for the new schedule-related fields. Next release will expose these fields via our existing schedule REST APIs.

Upgrading

As part of this release, we have finished upgrading our PostgreSQL databases to version 14 and have updated all tests to use that version as well. We recommend mirror node operators plan their migration to PostgreSQL 14 at their earliest convenience. More details on the upgrade process can be found in our database guide.

The migrations in this release are estimated to take up to 2 hours against a mainnet database. However, the migration that takes up the bulk of this time will only run if it needs to correct block numbers. This is only necessary if your mirror node is a partial mirror node and has an effective start date that occurs after the stream start.

v0.55

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: MAY 4, 2022
{% endhint %}
```

This release is mainly focused on finishing out our support for HIP-336 Approval and Allowance API for Tokens. We added support for the new CryptoDeleteAllowance transaction and removed support for the CryptoAdjustAllowance transaction that didn't make it into the final design. NFT allowances are tracked at the NFT transfer granularity allowing for up to date allowance information on the mirror node. Current spender information will show up in both `/api/v1/accounts/{id}/nfts` and `/api/v1/tokens/{id}/nfts` REST APIs. We also added the `isapproval` flag to APIs that show transfers.

With more developers using computers using Apple's M-series CPUs, it became clear the mirror node needed to support ARM-based architectures to accommodate them. In this release we added multi-architecture Docker images using `docker buildx`. We now push `linux/amd64` and `linux/arm64` variants to our Google Container Registry. If there's a need for additional operating systems or architectures in the future it can easily be expanded upon.

We also updated our GCP Marketplace application to the latest version.

v0.54

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: APRIL 25, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: APRIL 19, 2022
{% endhint %}
```

This release adds support for three new REST APIs and four HIPs.

HIP-21 describes the need for a free network info query to enable SDKs and other clients to be able to retrieve the current list of nodes. In v0.49.1, we added a new `NetworkService.getNodes()` gRPC API. In this release, we're adding an equivalent address book API to our REST API. In addition to the standard order and limit parameters, it supports a `file.id` query parameter to filter by the two address books `0.0.101` or `0.0.102` and a `node.id` query parameter to filter nodes and provide pagination.

GET `/api/v1/network/nodes`

```
{
  "nodes": [
    {
```

```

    "description": "",
    "fileid": "0.0.102",
    "memo": "0.0.3",
    "nodeaccountid": "0.0.3",
    "nodecerthash": "0x3334...",
    "nodeid": 0,
    "publickey": "0x308201...",
    "serviceendpoints": [
      {
        "ipaddressv4": "13.124.142.126",
        "port": 50211
      }
    ],
    "timestamp": {
      "from": "1636052707.740848001",
      "to": null
    }
  }
},
"links": {
  "next": null
}
}

```

HIP-336 describes new Hedera APIs to approve and exercise allowances to a delegate account. An allowance grants a spender the right to transfer a predetermined amount of the payer's hbars or tokens to another account of the spender's choice. In v0.50.0 we added database support to store the new allowance transactions. In this release, two new REST APIs were created to expose the hbar and fungible token allowances. Full allowance support won't be available until a future release when consensus nodes enable it on mainnet.

GET /api/v1/accounts/{accountId}/allowances/crypto

```

{
  "allowances": [
    {
      "amountgranted": 10,
      "owner": "0.0.1000",
      "spender": "0.0.8488",
      "timestamp": {
        "from": "1633466229.96874612",
        "to": "1633466568.31556926"
      }
    },
    {
      "amountgranted": 5,
      "owner": "0.0.1000",
      "spender": "0.0.9857",
      "timestamp": {
        "from": "1633466229.96874612",
        "to": null
      }
    }
  ],
  "links": {}
}

```

GET /api/v1/accounts/{accountId}/allowances/tokens

```
{
  "allowances": [
    {
      "amountgranted": 10,
      "owner": "0.0.1000",
      "spender": "0.0.8488",
      "tokenId": "0.0.1032",
      "timestamp": {
        "from": "1633466229.96874612",
        "to": "1633466568.31556926"
      }
    },
    {
      "amountgranted": 5,
      "owner": "0.0.1000",
      "spender": "0.0.9857",
      "tokenId": "0.0.1032",
      "timestamp": {
        "from": "1633466229.96874612",
        "to": null
      }
    }
  ],
  "links": {}
}
```

Also on the REST API, we added support for HIP-329 CREATE2 addresses. Now any API that accepts a contract ID will also accept the 20-byte EVM address as a hex-encoded string. We improved the performance of the REST API by adding cache control headers to enable distributed caching via a CDN. The performance of the list transactions by type REST API saw a fix to improve its performance.

As part of HIP-260, contract precompile call data now populates new fields amount, gas, and functionparameter inside ContractFunctionResult within the TransactionRecord. Mirror node now stores these fields and exposes them via its existing contract results REST APIs.

There were a number of security improvements made to containerized mirror nodes. All Docker images now run as non-root regardless of running in Kubernetes or Docker Compose. The helm charts saw changes to conform to the Kubernetes restricted pod security standard. This ensures the mirror node runs with security best practices and reduces its overall attack surface. The Kubernetes Pod Security Standard replaces the deprecated PodSecurityPolicy and as such we've removed all configuration related to the latter.

Upgrading

This release has a long migration that is expected to take around 75 minutes to complete, depending upon your database hardware and configuration. As always, we recommend a red/black deployment to eliminate downtime during migrations. If you're using the hedera-mirror-common chart, please check the kube-prometheus-stack upgrade notes to ensure Prometheus Operator can update successfully.

v0.53

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: APRIL 7, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: MARCH 29, 2022
{% endhint %}
```

This release is mainly focused around the area of data integrity and ensuring the data in the mirror node is consistent with consensus nodes. To this end, we added an errata database migration that only runs for mainnet and corrects three known issues that impacted the stream files. The state of the consensus nodes was never impacted, only the externalization of these changes to the stream files that the mirror node consumes.

To find the inconsistent data and ensure it stays consistent going forward, we added a new balance reconciliation job. This job runs nightly and compares the balance file information against the record file information to ensure they are in sync. It does three checks for each balance file: verifies the balance files add up to 50 billion hbars, verifies the aggregated hbar transfers match the balance file, and verifies the aggregated token transfers match the balance file. It can be disabled if not needed via `hedera.mirror.importer.reconciliation.enabled=false`.

We also fixed a bug that caused transfers with a zero amount to show up for crypto create transactions with a zero initial balance. This was due entirely to our code inserting the extra transfers, not because of any problem in the stream files. We also fixed an REST API bug that caused the contract byte code to show up as double encoded to hex.

For the Rosetta API, we added account alias support to various endpoints. And we now support parsing contract results for precompiled contract functions like HTS functions. This capability is disabled by a feature flag and will be enabled in a future release.

Upgrading

This release contains a couple medium sized database migrations to correct the erroneous data in the database. It is expected to take about 45 minutes against a full mainnet database.

v0.52

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: MARCH 18, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: MARCH 15, 2022
{% endhint %}
```

HIP-331 is a community contributed improvement proposal requesting the addition of a new REST API to retrieve an account's list of owned non-fungible tokens (NFTs). The mirror node has an existing `/api/v1/tokens/{tokenId}/nfts` API to retrieve all NFTs for a given token, but it didn't satisfy the requirement to show NFTs across token classes. This release adds the new `/api/v1/accounts/{accountId}/nfts` API to satisfy this need. It is our first API with multiple query parameters required for paging and as such has a few restrictions around their use. Please see the OpenAPI description for this API for further details.

```
GET /api/v1/accounts/0.0.1001/nfts?
token.id=gte:1500&serialnumber=gte:2&order=asc&limit=2
```

```
{
  "nfts": [
    {
      "accountid": "0.0.1001",
      "createdtimestamp": "1234567890.000000006",
      "deleted": false,
      "metadata": "bTI=",
```

```

    "modifiedtimestamp": "1234567890.0000000006",
    "serialnumber": 2,
    "tokenid": "0.0.1500"
  },
  {
    "accountid": "0.0.1001",
    "createdtimestamp": "1234567890.0000000008",
    "deleted": false,
    "metadata": "bTM=",
    "modifiedtimestamp": "1234567890.0000000008",
    "serialnumber": 3,
    "tokenid": "0.0.1500"
  }
],
"links": {
  "next": "/api/v1/accounts/0.0.1001/nfts?
order=asc&limit=2&token.id=gte:0.0.1500&serialnumber=gt:3"
}
}

```

The mirror node now has performance tests written using k6 for all of our APIs. These tests can be used to verify the performance doesn't regress from release to release. In the future, we plan to integrate these into a nightly regression test suite to improve our current approach of testing each release.

A number of deployment issues were addressed in this release. We now disable leader election by default until we can fix the issues with its implementation. Likewise we changed the importer Kubernetes deployment strategy from a rolling update to recreate to avoid ever having multiple importer pods running concurrently. A migration readiness probe was added to the importer. This will mark importer pods as unready until it completes all database migrations. Doing this will ensure Helm doesn't finish its release and run its tests before the migrations are completed.

We continue to fine tune our Rosetta implementation with a number of performance improvements and bug fixes. The performance of the Rosetta get genesis balance script was improved to reduce initial startup time. The embedded PostgreSQL container was upgraded to PostgreSQL 14. The Rosetta unified Docker image was updated to comply with the Rosetta persistence requirements.

v0.51

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: FEBRUARY 28, 2022
{% endhint %}

```

```

{% hint style="success" %}
TESTNET UPDATE COMPLETED: FEBRUARY 25, 2022
{% endhint %}

```

This is a smaller release focusing on observability improvements and Rosetta API fixes.

On the observability front, we've reduced the volume of log information the REST API produces in half. We also change the REST API to generate a consistent trace log for all responses that includes accurate client IPs, the elapsed time, and a status code. We reduced the number of time series by about 50% that the mirror node produces to reduce monitoring costs.

For the Rosetta API, we added a workaround for the missing disappearing token transfer issue that allows the check data reconciliation to pass. Overall reconciliation time was improved by tweaking configuration parameters and improving NFT balance tracking performance. We worked around slow genesis

will expose this information via a REST API as detailed in the design document.

Multiple new fields were added to the contract REST APIs as outlined in HIP-226 and HIP-227. The fields bloom, result, and status were added to the contract results API response. result and status show similar information with the former being the HAPI response enum while the latter returning 0x1 or 0x0 to show if the transaction was successful or not, as is common in web3 APIs. We also added bloom to the contract logs API response. Finally, we now return a partial response for contract calls without a result.

The importer component added a new `hedera.mirror.importer.parser.record.entity.persist.topics` property to control the persistence of topic messages. This can be set to false for mirror node operators if topic message data is not being used. On mainnet alone, this data currently takes up to 2TB worth of storage.

The Monitor component gained support for parallel node validation to improve startup performance. Now all validation is done in a background thread, adding and removing nodes as necessary while the publisher thread continues publishing transactions without any interruptions. This re-work also fixed issues with subscription halting during node validation and taking too long to validate a down node.

Rosetta saw a few important improvements including adding support for HIP-31 expected token decimals. The Rosetta unified Docker image saw functionality added to automatically restore the database using a database snapshot on initial startup.

Breaking Changes

As part of HIP-329 CREATE2, we renamed the existing `solidityaddress` in the contract REST API to `evmaddress`. This new name accurately reflects the naming in the HIP and protobuf and avoids tying the address to Solidity when Hedera supports more than just Solidity contracts.

v0.49

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: FEBRUARY 15, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: FEBRUARY 4, 2022
{% endhint %}
```

This release implements three Hedera Improvement Proposals (HIPs) and upgrades the mirror node database to the latest version.

HIP-21 describes the need for a free network info query to enable SDKs and other clients to be able to retrieve the current list of nodes. To satisfy this need we added a new `NetworkService.getNodes()` streaming gRPC API to get the list of current nodes from the address book network file. By making it a streaming API we avoid the client having to handle paging themselves, while still allowing us to split the large address book into smaller chunks. Since there are two address book files, we provide an option to choose which FileID to return.

```
message AddressBookQuery {
    .proto.FileID fileid = 1; // The ID of the address book file on the network.
    Can be either 0.0.101 or 0.0.102.
    int32 limit = 2;           // The maximum number of node addresses to
    receive before stopping. If not set or set to zero it will return all node
    addresses in the database.
}
```

```

service NetworkService {
    rpc getNodes (AddressBookQuery) returns (stream .proto.NodeAddress);
}

```

HIP-171 describes the need for returning the payer account in the topic message REST API response. This release does just that while also adding in the topic message chunk information that was present in the gRPC API but missing from the REST API.

```

{
+  "chunkinfo": {
+    "initialtransactionid": {
+      "accountid": "0.0.1000",
+      "nonce": 0,
+      "scheduled": false,
+      "transactionvalidstart": "1234567890-0000000006"
+    },
+    "number": 2,
+    "total": 5
+  },
+  "consensustimestamp": "1234567890.0000000001",
+  "topicid": "0.0.7",
+  "message": "bwVzc2FnZQ==",
+  "payeraccountid": "0.0.1000",
+  "runninghash": "cnVubmluZ19oYXNo",
+  "runninghashversion": 2,
+  "sequencenumber": 1
}

```

Continuing our support for HIP-32 auto account creation, we added alias support to our accounts REST API. Now when you query `/v1/api/accounts/:id` the id can be either a Hedera entity in the `0.0.x` form or a hex-encoded alias. An account's alias will now also show up in all of the accounts REST API output.

A lot of testing was done to ensure that PostgreSQL 14 functions correctly with the mirror node and provides as good as or better performance to older versions. We are now in the process of migrating our Hedera managed mirror nodes to PostgreSQL 14. We recommend other mirror node operators consider upgrading to the latest database version at their earliest convenience and have provided upgrade instructions to aid in that process.

v0.48

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: JANUARY 26, 2022
{% endhint %}

```

```

{% hint style="success" %}
TESTNET UPDATE COMPLETED: JANUARY 18, 2022
{% endhint %}

```

HIP-206 adds a parent consensus timestamp to the transaction record for internal transactions that occur like HTS precompiles invoked from a smart contract. To round out the nonce support in the last release we added `parentconsensustimestamp` to `/api/v1/accounts/{id}` and `/api/v1/transactions`. This field helps define the parent/child relationships between transactions.

HIP-226 describes the recently added contract results REST API. Each release we've been iterating and adding more functionality to the API until it matches the description in the HIP. This release adds the list of logs generated by a


```

operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/alertmanagerconfigs.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/alertmanagers.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/podmonitors.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/probes.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/prometheuses.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/prometheusrules.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/servicemonitors.yaml
kubectl apply --server-side -f https://raw.githubusercontent.com/prometheus-
operator/prometheus-operator/v0.53.1/example/prometheus-operator-crd/
monitoring.coreos.com/thanosrulers.yaml

```

v0.47

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: JANUARY 3, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE COMPLETED: DECEMBER 30, 2021
{% endhint %}

```

This release continues the focus on Smart Contracts 2.0. The mirror node is useful for debugging a smart contract execution and our focus has been on providing APIs to make developers' lives easier. To that end, we added support for transaction ID nonce, a new contract logs REST API, and a new web3 API component.

The new Web3 API module provides an implementation of existing JSON-RPC APIs for the Hedera network. JSON-RPC API is a widely used standard for interacting with distributed ledgers. The aim in providing a Hedera implementation of these APIs is to ease the migration of existing dApps to Hedera and simplify the developer on-ramp. Currently, the Web3 module only provides a partial implementation of the Ethereum JSON-RPC API. Specifically, only the `ethblockNumber` method has been implemented in this release as we focused on putting the groundwork in place first.

As part of HIP-32 and HIP-206 a nonce field was added to the TransactionID protobuf to guarantee uniqueness for platform generated transactions. This nonce field was added to any REST API that returns transaction data. A nonce query parameter was added to `/api/v1/transactions/:transactionId`, `/api/v1/transactions/:transactionId/stateproof`, and `/api/v1/contracts/results/:transactionId` to be able to distinguish between a user-submitted transaction and an internal transaction generated as a result of that transaction. Note that `/api/v1/transactions/:transactionId` without a nonce parameter will default to returning all transactions regardless of nonce while the other APIs will default nonce to 0.

The new `/api/v1/contracts/{id}/results/logs` REST API provides a search API to query for logs across contract executions for a particular contract. Searching by consensus timestamp and topics is supported. Note that for performance

v0.46

[illegible]

}

common module to share code with a future API module.

Database Migration

configuration flags.

v0.45.0

```
{% endhint %}
```

```
{% endhint %}
```

in the past.

Please take a look and let us know if you have any feedback.

columns for partitioning and co-locate them with other tables as appropriate.

reconciliation was also addressed.

v0.44.0

Making progress on transitioning our database to CitusDB, this release adds a new v2 schema with initial support for CitusDB. Automated testing against CitusDB was added to our CI pipeline so that it runs concurrently with the v1 PostgreSQL-based schema. The transaction payer account ID was added to transfer related tables. This will be used as a distribution column for database partitioning across a dimension that is not time-based. This allows the mirror node to scale reads and writes as more transaction payers use the system.

The rest of the release is mainly focused around performance improvements. We no longer persist minimal entity information for every entity ID encountered in a transaction. This was a performance drag but also caused problems with our plans to track entity history in an upcoming release. A few of our reference tables were removed in favor of using an application enum instead to map protobuf values to descriptive strings.

On the REST API, retrieval of accounts by public key was optimized to improve its performance. If your application does not require balance information, you can see additional performance gains by setting the new balance parameter to false for account API calls. The code was optimized to replace `Array.concat` with `Array.push` and to cache entity ID construction. The biggest change is probably the potentially breaking change to the limit parameter.

Breaking Changes

The maximum number of rows the REST API can return was changed from 500 to 100. Likewise the default number of rows the REST API returns if the limit parameter is unspecified was changed from 500 to 25. If a request is sent requesting more than 100 it won't fail. Instead, it will transparently use the smaller of the two values. As a result, this should not be a breaking change unless your application makes assumptions about the exact number of results being returned. We may tweak these values in the future for performance reasons so it's good practice to update your application to handle arbitrary limits and results.

v0.43.0

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: NOVEMBER 18, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: NOVEMBER 12, 2021
{% endhint %}
```

Smart Contracts

With Hedera's increased focus on Smart Contracts, we took the time to revamp the mirror node's smart contract support and lay the groundwork for future enhancements. As detailed in the design document, plans include new contract-specific REST APIs and Ethereum-compatible APIs in the future.

To prepare for that, the database schema and importer were updated to normalize and store all contract-related information, fixing long-standing bugs like not storing contract bytecode and child contracts. The contract table was split from the generic entity table and work was started on making all the entity tables maintain a history of all changes. The REST API now supports searching for and retrieving specific contracts. Below is an example of retrieving a contract:

```
/api/v1/contracts/{id}
```

```
{
```

```

    "adminkey": {
      "type": "ProtobufEncoded",
      "key": "7b2233222c2233222c2233227d"
    },
    "autorenewperiod": 7776000,
    "bytecode": "0xc896c66db6d98784cc03807640f3dfd41ac3a48c",
    "contractid": "0.0.10001",
    "createdtimestamp": "1633466229.96874612",
    "deleted": false,
    "expirationtimestamp": "1633466229.96874612",
    "fileid": "0.0.1000",
    "memo": "First contract",
    "obtainerid": "0.0.101",
    "proxyaccountid": "0.0.100",
    "solidityaddress": "0x000000000000000000000000000000000000000000000000000000000000003E9",
    "timestamp": {
      "from": "1633466229.96874612",
      "to": "1633466568.31556926"
    }
  }
}

```

Data Architecture

Over the last few months, work has been underway to analyze possible PostgreSQL replacements as the need for handling an ever-increasing amount of data puts strain on the existing mirror node database. After agreeing upon the acceptance criteria, priority was placed on a PostgreSQL-compatible distributed database that can shard our time-series data across many nodes for scale-out reads and writes. That would ensure the quickest time to market and ease transition for Hedera and others using the open source mirror node software. The four distributed databases we chose for our proof of concept included CitusDB, CockroachDB, TimescaleDB, and YugabyteDB.

After a detailed analysis of each, we chose CitusDB for our next database due to its excellent PostgreSQL compatibility (it's a PostgreSQL extension) and its mature support for sharding time-series data. Its distributed query engine routes and parallelizes DDL, DML, and other operations on distributed tables across the cluster. And its columnar storage can compress data up to 8x, speeds up table scans, and supports fast projections. This release contains some foundational work to get our schema ready for partitioning. You can track our progress as we work towards integrating CitusDB into our codebase over the next few months. We plan on maintaining support for both databases for a period of time after the work is complete.

Performance Improvements

As is usually the case, we took the time to optimize various pieces of the system to work at scale. Our transactions REST API saw some performance improvements by rewriting them using Common Table Expressions (CTE). This will pay future dividends with CitusDB as it allows queries to be ran in parallel easier. An issue with `/api/v1/topics/{id}/messages` timing out for some topics was addressed and the realm and topic number columns were combined to reduce the table and index size. `/api/v1/tokens/{id}/balances` also saw some performance improvements that decreased its average response time. Configuration options for faster historical ingestion were documented so that mirror node operators can get historical data faster.

v0.42.0

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: OCTOBER 22, 2021
{% endhint %}

```



```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: OCTOBER 18, 2021
{% endhint %}
```

This release saw a lot of improvements to the mirror node's Hedera Token Service functionality. Support for HIP-24 pause feature on Hedera Token Service was completed. The importer can ingest the new token pause and unpause transaction types and update the token appropriately. Likewise, the token REST API was updated to show the new pause key and pause status.

Along those lines, the token REST API was also updated to show the token memo and a flag to show if it's deleted. Now when an account is dissociated from a token its supply will be properly updated to show the negative transfer. And if the token in that dissociate is of type NFT, all of the NFTs owned by that account will be properly marked as deleted. We also fixed issues with some special negative transfer amounts showing up in the transactions REST API.

A new network supply REST API was added to show the released supply. Having the open source mirror node calculate and show the release supply avoids any single point of failure with the current system because a user could ask multiple mirror nodes and compare their answers (or run their own mirror node).

GET /api/v1/network/supply

```
{
  "timestamp": "123456870.854775807",
  "releasedsupply": 18000000000000000000,
  "totalsupply": 50000000000000000000
}
```

Continuing our theme of improving the Rosetta API, NFT support was added to the data and construction APIs. We took the time to convert it to a standard configuration library and reorganize the package structure to be flatter and more consistent. And contexts were added to every layer to enable proper cancellation and timeout support.

v0.41.0

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: OCTOBER 6, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: SEPTEMBER 30, 2021
{% endhint %}
```

This release focuses our efforts on improving our Rosetta API and making it ready for production use. A new Rosetta Helm chart was added for production deployments to Kubernetes. Observability improvements include health probes, metrics, request logs, alerts, and a Grafana dashboard. Postman integration tests were added to verify post-deployment functionality. Finally, a few important bugs were fixed including missing peer IP addresses and a token balance reconciliation failure.

The importer component was optimized to ingest transactions at 15,000 TPS or higher. This change included improvements to reduce CPU and memory usage while simultaneously increasing the allocated memory available to the process.

Other enhancements include revalidating main nodes periodically in the monitor and adding TLS support for the REST API's database connection.

v0.40.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: SEPTEMBER 27, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: SEPTEMBER 16, 2021  
{% endhint %}
```

This release adds support for HIP-23 Automatic Token Association. This feature allows users to opt-in to receiving fungible or non-fungible tokens automatically as part of a transfer without having to be previously associated with the token. The mirror node now stores these implicitly created associations and returns them via its REST API. Additionally, we show the `maxautomatictokenassociations` in the accounts REST API.

Besides updating it for HIP-23, the REST API saw quite a few other fixes and improvements. The accounts API now displays its memo and the `receiverSigRequired` field. The REST API packages were renamed to use the `@hashgraph` NPM package scope. This shouldn't be a breaking change as we don't currently publish those packages to NPM. A number of APIs were fixed to ensure lists were returned in a deterministic sort order. Also, the OpenAPI specification was fixed up so that it accurately reflects the current API and can be used to generate client code. Finally, the schedules API had some performance improvements.

On the monitoring side, we enhanced our Grafana dashboards to make them compatible with Grafana Cloud by adding datasource and cluster drop-downs.

v0.39.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: AUGUST 31, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 30, 2021  
{% endhint %}
```

This release provides compatibility with Hedera Services 0.17 including support for Non-Fungible Tokens (NFTs) and its enhancement to custom fees. For the latter, an NFT creator can set a royalty fee to be charged when fungible value is exchanged for one of their creations and the mirror node has been updated to track this new type of custom fees. Support was also added for effective payer accounts in assessed custom fees and for storing net-of-transfers in fractional fees.

The mostly unused data generator module was removed, resulting in a large increase in code coverage. Coverage has increased from 84% to 92%.

A good amount of bugs were fixed including a crash on REST API startup if the database was down, monitor taking too long to startup, OpenAPI fixes, and more.

v0.38.1

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: AUGUST 17, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 17, 2021  
{% endhint %}
```

This release is a small bug fix release that contains some important fixes for our mirror node monitoring component. We added a new cluster health check to the

monitor that takes into account publishing status. The load balancer uses this health check to determine which cluster to route traffic to. The old health check endpoint didn't take into account whether transaction publishing was active or successful and so would not route traffic to the public mirror node during main node upgrades.

Besides the new health check, the monitor had fixes to its rate calculation at low TPS, not sampling when idle, node validation, and the alerts it generates. The mainnet network configuration of the monitor now points to the public mirror node and we've added the new previewnet node to the previewnet network configuration.

There were also a number of other fixes to clean up code and fix tests. We've made an effort to reduce our code smells as seen in SonarCloud.

v0.38.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: AUGUST 16, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 13, 2021  
{% endhint %}
```

This release wraps up NFT and custom fee support by adding additional test coverage and fixing any remaining bugs. Specifically, NFT support was added to our monitor tool and our acceptance tests. Custom fees was also added to the acceptance tests and had some bug fixes.

Mainnet public saw some monitoring improvements including adding HTTPS support to our external monitor dashboard and the addition of a platform not active alert that inhibits all other alerts.

There were a number of bug fixes in this release. The stream file health check that was disabled in the last release due to a bug was fixed and re-enabled. The address book update flow saw a couple of important fixes as well.

Breaking Changes

The payer account ID in transaction assessed custom fee REST API response was removed. This is a change in services 0.16 whereby custom fees are now charged from the account who send the triggering tokens, not necessarily the payer of the transaction.

v0.37.2

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: AUGUST 4, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 5, 2021  
{% endhint %}
```

A small bug fix release that addresses some issues with our HIP-18 support.

v0.37.1

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JULY 29, 2021  
{% endhint %}
```

```
{% hint style="success" %}
```

TESTNET UPDATE COMPLETED: JULY 15, 2021
{% endhint %}

This release broadens our support for non-fungible tokens (NFTs) with new NFT-specific REST APIs. A new API was added to return a list of NFTs for a particular token ID. We also added a new API to return a single NFT by its token ID and serial number. Finally, we added an API to see the transaction history for a particular NFT. In an effort to have more manageable REST API code, we now adopt a more object-oriented approach by utilizing models, view-models and services. Below is an example of the three new APIs:

GET /api/v1/tokens/0.0.1500/nfts

```
{
  "nfts": [{
    "accountid": "0.0.1002",
    "createdtimestamp": "1234567890.000000010",
    "deleted": false,
    "metadata": "ahf=",
    "modifiedtimestamp": "1234567890.000000010",
    "serialnumber": 2,
    "tokenid": "0.0.1500"
  }, {
    "accountid": "0.0.1001",
    "createdtimestamp": "1234567890.000000009",
    "deleted": false,
    "metadata": "bTM=",
    "modifiedtimestamp": "1234567890.000000008",
    "serialnumber": 1,
    "tokenid": "0.0.1500"
  }],
  "links": {
    "next": null
  }
}
```

GET /api/v1/tokens/0.0.1500/nfts/1

```
{
  "accountid": "0.0.1001",
  "createdtimestamp": "1234567890.000000008",
  "deleted": false,
  "metadata": "bTM=",
  "modifiedtimestamp": "1234567890.000000009",
  "serialnumber": 1,
  "tokenid": "0.0.1500"
}
```

GET /api/v1/tokens/0.0.1500/nfts/1/transactions

```
{
  "transactions": [{
    "consensustimestamp": "1234567890.000000009",
    "transactionid": "0.0.8-1234567890-000000009",
    "receiveraccountid": "0.0.1001",
    "senderaccountid": "0.0.2001",
    "type": "CRYPTOTRANSFER"
  }, {
    "consensustimestamp": "1234567890.000000008",

```

```

    "transactionid": "0.0.8-1234567890-0000000008",
    "receiveraccountid": "0.0.2001",
    "senderaccountid": null,
    "type": "TOKENMINT"
  }],
  "links": {
    "next": null
  }
}

```

v0.36.0

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: JULY 19, 2021
{% endhint %}

```

We are happy to announce the availability of a publicly accessible, free-to-use, mainnet Mirror Node operated by the Hedera team. As part of this, we put a large amount of effort into fine-tuning our Kubernetes deployment. We migrated to Flux 2, a GitOps-based deployment tool that allows us to declaratively specify the expected state of the Mirror Node in git and manage our rollouts. You can browse our deploy branch and see the exact config and versions rolled out to various clusters and environments. The Helm chart was updated to add PodDisruptionBudgets, adjust alert rules and other improvements to make it easier to automate the deployment.

This release is the first version of the Mirror Node with preliminary support for non-fungible tokens (NFTs). NFT support is being added to the Hedera nodes as outlined in HIP 17. We spent time designing how that NFT support will look like for the Mirror Node. Modifications to the schema were made to add new tables and fields and the Importer was updated to ingest NFT transactions. The existing REST APIs were updated to add NFT related fields to the response. This includes adding a type field to the token related APIs to indicate fungibility and anfttransfers to /api/v1/transactions/{id}:

```

{
  "transactions": [{
    "consensustimestamp": "1234567890.000000001",
    "name": "CRYPTOTransfer",
    "nfttransfers": [
      {
        "receiveraccountid": "0.0.121",
        "senderaccountid": "0.0.122",
        "serialnumber": 104,
        "tokenid": "0.0.14873"
      }
    ]
  }]
}

```

One thing to note is that we did not add NFT transfers to the list transactions endpoint in an effort to reduce the size and improve the performance of that endpoint. In the next release, we will add new NFT specific REST APIs.

Continuing upon the theme of the last release, we made additional changes to the Rosetta API to bring it up to par with the rest of the components. Rosetta now includes support for HTS via both its data and construction APIs.

The Importer saw a large focus on improving performance and resiliency. It is now highly available (HA) when run inside Kubernetes. This allows more than one instance to run at a time and to failover to the secondary instance when the

primary becomes unhealthy. A special Kubernetes ConfigMap named `leaders` is used to atomically elect the leader.

We're improving our ingestion time dramatically for entity creation. Previously those were database finds followed by updates. Since inserts are always faster than find and updates, we've optimized this to insert the updates into a temporary table and at the end upsert those to the final table. A record file with 6,000 new entities went from 21 seconds to 600 ms, making it 35x improvement. Balance file processing was optimized to greatly reduce memory by only keeping one file in memory at a time.

Breaking Changes

In honor of Juneteenth and as part of the general industry-wide movement, we renamed our master branch to `main`. If you have a clone or fork of the Mirror Node Git repository, you will need to take the below steps to update it to use `main`:

```
git branch -m master main
git fetch origin
git branch -u origin/main main
git remote set-head origin -a
```

As part of our optimization to reduce memory usage, we now process some things earlier in the lifecycle. Due to this we had to rename some properties to reflect this change. We also changed the disk structure if you are using the `keepFiles` (now renamed to `writeFiles`) properties to write the stream files to disk after download. It is no longer archived into folders by day. Instead, the folder structure will exactly match the structure in the bucket. This opens the possibility for a mirror node to download and mirror the bucket itself using a S3 compatible API like MinIO. Below is a summary of the renamed properties:

```
Renamed hedera.mirror.importer.downloader.balance.keepSignatures to
hedera.mirror.importer.downloader.balance.writeSignatures
Renamed hedera.mirror.importer.parser.balance.keepFiles to
hedera.mirror.importer.downloader.balance.writeFiles
Renamed hedera.mirror.importer.parser.balance.persistBytes to
hedera.mirror.importer.downloader.balance.persistBytes
Renamed hedera.mirror.importer.downloader.event.keepSignatures to
hedera.mirror.importer.downloader.event.writeSignatures
Renamed hedera.mirror.importer.parser.event.keepFiles to
hedera.mirror.importer.downloader.event.writeFiles
Renamed hedera.mirror.importer.parser.event.persistBytes to
hedera.mirror.importer.downloader.event.persistBytes
Renamed hedera.mirror.importer.downloader.record.keepSignatures to
hedera.mirror.importer.downloader.record.writeSignatures
Renamed hedera.mirror.importer.parser.record.keepFiles to
hedera.mirror.importer.downloader.record.writeFiles
Renamed hedera.mirror.importer.parser.record.persistBytes to
hedera.mirror.importer.downloader.record.persistBytes
```

v0.35.0

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: JULY 8, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: JUNE 21, 2021
{% endhint %}
```

Most of the changes in this release were operational improvements around our

Kubernetes deployment. These changes were necessary as we begin to convert more environments from virtual machines to Kubernetes-based. We added our acceptance tests to the Helm chart so that it can trigger automatically during upgrades and verify the deployment was successful. On the importer, we added a new health check to the probes that verifies that stream files are successfully being parsed. And we fixed the importer so that the probes are started before long-running database migrations, allowing us to finally enable its liveness probe. There were a lot of smaller fixes to the charts, so please see the linked PRs for further details.

The monitor saw a brand new REST API that lists active subscriptions. This is used in our cluster to determine overall cluster health and route traffic via our load balancers. We added an OpenAPI spec and Swagger UI for this API as well.

Special thanks to @si618 for fixing the build on Windows and adding a GitHub workflow to make sure it stays fixed.

Breaking changes

The REST API maximum and default limit was lowered from 1000 to 500. If you explicitly send a number of more than 500, your request will fail. Please update your client code appropriately.

v0.34.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JUNE 16, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JUNE 11, 2021  
{% endhint %}
```

In Hedera Mirror Node v0.34.0, we started work on designing support for NFTs that will come in a future Hedera Services release.

By default, the mirror node will validate that at least one-third of all nodes in the address book have signed a stream file before importing it into its database. This ensures that the main nodes have reached two-thirds consensus on the transactions in the file. For performance or verification reasons, you may want to decrease or increase this default percentage. To support this use case, we added a `hedera.mirror.importer.downloader.consensusRatio` property that controls the ratio of verified nodes (nodes used to come to consensus on the signature file hash) to the total number of nodes available.

We took the time to undertake some major dependency upgrades for the Rosetta API. This included major updates to the Hedera and Rosetta SDKs that both required a large amount of refactoring. A number of bugs in Rosetta were addressed as well as improvements to Rosetta's CI workflow. These changes lay the groundwork for additional Rosetta improvements in a future release.

To avoid duplication, we wanted to unify our JMeter and Monitor performance tests. To do so, we needed the newer monitor tool to have feature parity with our JMeter tests. To accomplish this, we've split the publish to HAPI and subscribe to mirror node flows in the monitor to allow for subscribe only. In this iteration, only the gRPC API supports subscribe only. With this change, we were able to remove our JMeter code and optimize the `hedera-mirror-test` image from 1.5G to 0.5G.

We made some operational improvements to our helm chart including alert dependencies. Alert dependencies help avoid a flood of alerts that are all related to the same root cause. We also made some bug fixes to the chart that could occur when enabling or disabling some components in favor of external

databases or message buses.

v0.33.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JUNE 10, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 21, 2021  
{% endhint %}
```

This release adds support for HAPI 0.13.2. This brings with it a new address book file format that is more compact and doesn't duplicate IP address and port information. We took the time to adjust our database to reflect the newer format while maintaining compatibility with the older format.

A big focus of this release was on improving the Helm charts for use in production deployments. We now auto-generate passwords for components that require one and ensure they remain the same on upgrades by using Helm's lookup feature. We added `env`, `envFrom`, `volumes`, `volumeMounts` properties to all charts for more flexible configuration. We added a `global.image.tag` chart property to make it easier to test out custom versions. And we made it easier to use dependencies that can be outside the cluster like Redis and PostgreSQL.

Some internal improvements saw us automating our release process so that version bumps and release note generation can be kicked off via GitHub. This now also includes generating a CHANGELOG and keeping it up to date with the release notes. And finally we updated our acceptance tests to automatically pull and use the latest address book along with validating all nodes to ensure only the latest, valid nodes are used for validation.

v0.32.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MAY 19, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 11, 2021  
{% endhint %}
```

In this release we took the time to do some performance optimizations of both the importer and the monitor. If you're using a containerized mirror node, the Java applications now uses more of the available memory that's already been allocated to it. We optimized the size of some internal queues to reduce the likelihood of out of memory errors. And we now use a more efficient streaming method to write entities to the database and avoid large memory allocations. All these combine to greatly reducing overall memory usage and improve overall performance for the importer. The monitor also saw performance improvements to allow it to publish transactions at a rate of 10,000 TPS.

This release updates more of our system to handle the revised scheduled transaction design that will be available soon on mainnet. Both the acceptance tests and monitor were updated to be able to publish the new transactions.

We now expose the raw transaction bytes encoded in Base64 format in the REST API. Persisting the bytes of the Transaction protobuf in the database is an option that's been available for a while but until now has not been available via the API. Persisting the data is off by default as does increase the size of the database quite a bit. The Hedera managed mirror nodes will not have that functionality turned on to reduce storage.

v0.31.0


```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: APRIL 30, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: APRIL 26, 2021
{% endhint %}
```

After scheduled transactions were made available in previewnet, we listened to user feedback and further iterated on the design to make it easier to use. This release adds support for this revised scheduled transactions design planned to be released in HAPI v0.13. There was no impact to our REST API format, only the importer needed to be updated to parse and ingest the new proto format. Our monitor API and acceptance tests will be adjusted in the next release once the SDKs add support for the new design.

This release also adds support for the newly announced account balance file format that was released in HAPI v0.12. The new protobufbased format will eventually replace the CSV format in July 2021. Until then, both formats will exist simultaneously in the bucket so users can transition at their leisure. Besides being more efficient to parse, the new files are also compressed using Gzip for reduced storage and download costs. We also took the time to improve the balance file parsing performance regardless of format. Average parse times should decrease by about 27%.

For our REST API, we now expose an entityid field on our transactions related APIs. This field represents the main entity associated with that transaction type. For example, if it was a HCS transaction it would be the topic ID created, updated, or deleted.

```
GET /api/v1/transactions/0.0.1009-1234567890-999999998
```

```
{
  "transactions": [{
    "consensustimestamp": "1234567890.999999999",
    "entityid": "0.0.108763",
    "validstarttimestamp": "1234567890.999999998",
    "chargedtxfee": 0,
    "memobase64": null,
    "result": "SUCCESS",
    "scheduled": false,
    "transactionhash": "aGFzaA==",
    "name": "CRYPTOUPDATEACCOUNT",
    "node": "0.0.3",
    "transactionid": "0.0.1009-1234567890-999999998",
    "validdurationseconds": "11",
    "maxfee": "33",
    "transfers": []
  }]
}
```

We continue to make progress towards our goal of switching to TimescaleDB. We fixed the user and database initialization issues and tested a migration from PostgreSQL. We switched out the TimescaleDB Helm chart to a more stable one and explored our hosting options for production. Finally, we switched to SCRAM-SHA-256 to improve the security of our database user authentication.

Breaking changes:

There were a number of breaking changes this release to be aware of. If you're using our Helm chart, we have switched the importer from a StatefulSet to a

Deployment since it no longer has the need for a persistent volume. We also switched the Traefik dependency from a Deployment to a DaemonSet. Both of these will require manual intervention to delete the old workload before upgrading. Support for Helm 2 was dropped since it is no longer supported by the community after November 13, 2020. If you're directly reading from our database, a rename of the tentities table and its columns may impact you as well.

v0.30.0

Mirror node v0.30 brings operational improvements with changes to our continuous integration and monitoring components.

With this release, we've completed the migration from CircleCI to GitHub Actions. CircleCI had some limitations with our use of Testcontainers for unit testing against 3rd party dependencies. We previously had a mixture of GitHub Actions and CircleCI with the latter using slightly different commands than local testing. Consolidating to GitHub Actions allowed us to reduce this difference and further parallelize our checks.

To improve our runtime observability and testing coverage, we've continued to invest in our monitor tool this cycle. Scheduled transaction support was recently added supporting both ScheduleCreate and ScheduleSign operations. We've added the three new mainnet nodes the monitor's default configuration. A bug with the monitor unable to reach expected TPS with multiple scenarios was fixed.

The REST API also saw some bug fixes including a fix to queries with a credit/debit parameter now able to retrieve token only transfers. The transaction API now populates the token transfers list for all transaction types instead of being limited to just crypto transfers.

v0.29.1

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: APRIL 5, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: MARCH 26, 2021
{% endhint %}
```

This release brings an assortment of under the hood improvements across modules and refinements of multiple REST API's.

Historical entity information prior to OA is now available. In this release we've added a repeatable Java migration that will import entity information from a mainnet network snapshot. This runs during upgrade, is configureable (hedera.mirror.importer.importHistoricalAccountInfo) and works in combination with the hedera.mirror.importer.startDatesetting.

The REST API now expands its filtering options support specifically around transfers and in relation to tokens. Previously the account.idand credit/debit filtering options supported HBAR transfers only, this release expands both filters to include tokens also.

The stateproof REST API and check-state-proof package have also been improved. The API now supports filtering for scheduled transactions via /api/v1/transactions/:transactionId/stateproof?scheduled=true as-well as a more compact response format. For record streams that utilize the newer improved HAPI v5 version the stateproof API response send back metadata hashes instead of the full raw bytes. With this, the response is more light weight.

```
{
  "addressbooks": [
```

```

    "address book content"
  ],
  "recordfile": {
    "head": "content of the head",
    "startrunninghashobject": "content of the start running hash object",
    "hashesbefore": [
      "hash of the 1st record stream object",
      "hash of the 2nd record stream object",
      "hash of the (m-1)th record stream object"
    ],
    "recordstreamobject": "content of the mth record stream object",
    "hashesafter": [
      "hash of the (m+1)th record stream object",
      "hash of the (m+2)th record stream object",
      "hash of the nth record stream object"
    ],
    "endrunninghashobject": "content of the end running hash object",
  },
  "signaturefiles": {
    "0.0.3": "signature file content of node 0.0.3",
    "0.0.4": "signature file content of node 0.0.4",
    "0.0.n": "signature file content of node 0.0.n"
  },
  "version": 5
}

```

The REST API now also supports repeatable account.id query parameters when filtering, with a configureable setting for the maximum number of repeated query parameters\

/api/v1/(accounts|balances|transactions)?account.id=:id&account.id=:id2...

GET /api/v1/accounts?account.id=0.0.7&account.id=0.0.9

```

{
  "accounts": [
    {
      "balance": {
        "timestamp": "0.000002345",
        "balance": 70,
        "tokens": [
          {
            "tokenid": "0.0.100001",
            "balance": 7
          },
          {
            "tokenid": "0.0.100002",
            "balance": 77
          }
        ]
      },
      "account": "0.0.7",
      "expirytimestamp": null,
      "autorenewperiod": null,
      "key": null,
      "deleted": false
    },
    {
      "balance": {
        "timestamp": "0.000002345",
        "balance": 90,
        "tokens": []
      },

```

```

        "account": "0.0.9",
        "expirytimestamp": null,
        "autorenewperiod": null,
        "key": null,
        "deleted": false
    }
},
"links": {
    "next": null
}
}

```

Multiple modules have also seen security and standardization improvements by the addition of more robust automated analysis tools such as gosec as-well as the implementation of suggestions from a 3rd party code audit.

This release also saw a step to support the new and improved v2 offerings of the (Java SDK)\[<https://github.com/hashgraph/hedera-sdk-java>\]. Both the monitor module and acceptance tests were updated to use the new SDK and utilize features such as in-built retry and support for scheduled transactions.

v0.28.2

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: MARCH 17, 2021
{% endhint %}

```

```

{% hint style="success" %}
TESTNET UPDATE COMPLETED: MARCH 10, 2021
{% endhint %}

```

This releases finalizes support for scheduled transactions and HAPI protobuf v0.12. Two new schedule specific REST APIs were added including /api/v1/schedules and /api/v1/schedules/:id. The former lists all schedules with various filtering options available and the latter returns a specific schedule by its schedule ID.

```

GET /api/v1/schedules?
account.id=0.0.1024&schedule.id=gte:4000&order=desc&limit=10

```

```

{
  "schedules": [
    {
      "adminkey": {
        "type": "ProtobufEncoded",
        "key": "7b2233222c2233222c2233227d"
      },
      "consensustimestamp": "1234567890.000030003",
      "creatoraccountid": "0.0.1024",
      "executedtimestamp": null,
      "memo": "Created per governing council decision dated 02/03/21",
      "payeraccountid": "0.0.1024",
      "scheduleid": "0.0.4000",
      "signatures": [
        {
          "consensustimestamp": "1234567890.000030001",
          "publickeyprefix": "CQkJ",
          "signature": "CQkJ"
        }
      ],
      "transactionbody": "AQECAgMD"
    }
  ]
}

```

```

    ],
    "links": {
      "next": null
    }
  }
}

```

In HAPI v0.12, new memo fields were added to all entity types bringing parity across all services. Mirror node now supports the new fields including for update operations where the memo field can be set to null, empty string or a non-empty string to keep, clear or replace the existing memo, respectively.

Historically, the importer application has always downloaded stream files and saved to the filesystem in one thread then read those files and ingested them into the database in another thread. This has sometimes caused the database and filesystem to get out of sync and require manual intervention to fix. It also makes the importer stateful and as a result could not support running multiple instances for high availability.

With this release, we've removed the need for importer to read and write to the filesystem. Instead, the downloader and parser threads now communicate via an in-memory queue. To accomplish this, we also had to remove the `applicationstatus` table in favor of calculating the last successful file directly from the stream file tables. In addition to fixing the aforementioned issues, the removal of the filesystem has resulted in a 5% latency improvement.

Other changes include adding an index field to `recordfile` table to simulate a blockchain index and updating our Google Marketplace to v0.27. Also, we added support for the v5 stream files to the check-state-proof client app.

0.27.0

```

{% hint style="success" %}
MAINNET UPDATE COMPLETED: FEBRUARY 22, 2021
{% endhint %}

```

```

{% hint style="success" %}
TESTNET UPDATE COMPLETED: FEBRUARY 11, 2021
{% endhint %}

```

This release adds a new REST API component that implements the Rosetta API. The Rosetta API is an open standard for integrating with blockchain-oriented systems. Implementing the Rosetta AP provides a number of advantages. It reduces the time and effort it takes for wallets, exchanges, etc. to integrate with the Hedera network if they have integrated with Rosetta in the past. Even if the systems integrator has not used Rosetta previously, using the Rosetta API in lieu of our separate REST API might be useful to reduce the friction with using a non-blockchain DLT like Hedera.

Scheduled transactions is an new feature being added to the main nodes in a future release. Scheduled transactions allows transactions to be submitted without all the necessary signatures and will execute once all the required parties sign. The mirror node has been updated to understand and store these new types of transactions. Additionally, we've updated our existing REST APIs to expose this information. The next release will add additional schedule specific REST APIs.

We made a concerted effort this release to improve our tests. Most of our flaky tests were fixed so that our continuous integration runs smoother. We also improved the stability of our acceptance tests. The REST API monitor also had some logging and useability fixes to aid in production observability.

v0.26.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: FEBRUARY 1, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JANUARY 22, 2021  
{% endhint %}
```

This release is mainly focused on adding support for the upcoming features in the main nodes. We added support for the newTotalSupply field to the transaction record in HAPI 0.10.0. We also documented our design for the upcoming scheduled transactions services that's coming in a future release of HAPI. Our next minor version will have preliminary support for that.

But by far the biggest change is support for the new record file V5 and signature file V5 format. These files are uploaded to cloud storage and pulled by the mirror nodes to populate its database. Since it's the core communication format between the main nodes and mirror nodes, it took a bit of refactoring and new code to support the new format while retaining compatibility with previous stream files.

Warning! If you don't upgrade your Mirror Node to v0.26.0 or later before HAPI v0.11.0 is released in a few weeks, your mirror node will be unable to process new transactions.

We continued our progress on switching to TimescaleDB. We integrated a TimescaleDB helm chart into our Kubernetes deployment and added migration scripts to convert from PostgreSQL to TimescaleDB. We're still in the testing phase so it's still recommended to stick with the v1 schema (the default) for now.

v0.25.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JANUARY 12, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JANUARY 8, 2021  
{% endhint %}
```

This release saw a slew of enhancements to our new monitoring module. The monitor is a standalone component that can publish and subscribe to transactions from various Hedera APIs to gauge the health of the system. New in this release is the ability to automatically create entities on startup using a new expression syntax. This is useful to avoid boilerplate configuration and manual entity creation steps that vary per environment.

A sample percentage property was added to the monitor to control how often the REST API should be verified. We took the time to properly document the monitor tool detailing its configuration and operational steps. Finally, we added a number of new metrics and a Grafana dashboard to view them.

We made progress towards our goal of replacing PostgreSQL with TimescaleDB. This release contains the initial database migrations to setup the mirror node from scratch using TimescaleDB. These migrations are hidden behind a feature flag. In an upcoming release we'll add further functionality including data migration scripts and Helm support.

v0.24.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: DECEMBER 28, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: DECEMBER 10, 2020  
{% endhint %}
```

This release adds OpenAPI 3.0 specification support to our REST API. The OpenAPI specification is available at `/api/v1/openapi.yaml` and serves as a formal specification of our API. Clients can use the specification to shorten the amount of time it takes to integrate with our API by generating code or tests harnesses. It also provides us with a new auto-generated API documentation site viewable at `/api/v1/docs`.

We now have support for the AWS Default Credential Provider Chain. Now instead of only being able to provide static access and secret keys in the configuration, you can rely on the default provider chain to retrieve your credentials automatically from the environment (environment variables, `/.aws/credentials`, etc). See our documentation for more information.

We've enhanced our monitoring tools to provide greater observability into the mirror node's operation. In addition to publishing, our monitor tool now supports subscribing to the gRPC and REST APIs to verify end to end functionality of Hedera. It will also generate metrics off this information. We take advantage of Loki's new log alerting capability and now can alert off of any errors we see in logs that might be cause for concern.

v0.23.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: DECEMBER 2, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 20, 2020  
{% endhint %}
```

This release focuses on finalizing our support for the new Hedera Token Service (HTS) provided by the Hedera API v0.9.0. There are no new HTS features, just various fixes to make it compatible with the latest protobuf. HTS is currently enabled in previewnet and should be enabled in testnet very soon, so please try it out and let us know if you have any feedback.

A new Helm chart was added to run the monitor application. The monitor is still under heavy development so stay tuned.

Most of the other changes were bug fixes. We now use SonarCloud to scan for vulnerabilities and bugs and have addressed all the major items. You can view our SonarCloud dashboard to track our progress. Entities are now only inserted for successful transactions and we fixed the wrong address book being updated. There were multiple issues with the state proof alpha API that were resolved. For the gRPC API, we improved its performance and lowered its CPU usage. Also related to gRPC, we now enable server sent keep alive messages and permit a lower client sent keep alive messages of 90 seconds, which should hopefully address timeout issues that some users have reported.

v0.22.0

This release continues our improvements to our Kubernetes support as well as monitoring and performance improvements across the modules.

We improved our custom PrometheusRule alerts for the Importer, gRPC and REST API modules, as well as added dashboards for our gRPC and REST API modules. Additionally, we increased our pod resources limits to optimize Importer ingestion and gRPC streaming performance in a Kubernetes cluster. Our existing js based monitor and REST performance tests were both updated to include HTS

support.

We also improved our data generator module with support for the majority of HAPI transactions the mirror node importer ingests. Additionally, we also added a java based monitor module that supports generation and publishing of transactions.

This release also includes an improvements to avoid the stale account info bug that stems from balance stream files being received at a slower frequency than record stream files. Now account creations and account info changes will be reflected in REST API call even though the updated balance may not have been received.\

We also extended our REST API support to include case insensitive support query parameters. /api/v1/transactions?transactionType=tokentransfers and /api/v1/transactions?transactiontype=tokentransfers are now both acceptable.

v0.21.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: NOVEMBER 24, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 13, 2020  
{% endhint %}
```

This release continues our focus on the Hedera Token Service (HTS) by adding three new token REST APIs. A token discovery REST API /api/v1/tokens shows available tokens on the network. A token REST API /api/v1/tokens/\${tokenId} shows details for a token on the network. A token supply distribution REST API /api/v1/tokens/\${tokenId}/balances shows token distribution across accounts. These APIs have already made their way to previewnet so check them out!

Continuing our HTS theme, we enhanced our testing frameworks with token support. Our acceptance tests can send HTS transactions to HAPI and wait for those transactions to show up via the mirror node REST API. Additionally, our performance tests can simulate a HTS load to test how the system responds to HTS transactions.

We improved our existing REST APIs by adding a way to filter by transaction type. When searching for transactions or showing the transactions for a particular account you can now filter via an optional transactionType query parameter. This feature can be used with the transactions API in the format /api/v1/transactions?transactionType=tokentransfers while the format for the accounts API is /api/v1/accounts/0.0.8?transactionType=TOKENTRANSFERS.

We improved our Kubernetes support with AlertManager integration. There are now custom PrometheusRule alerts for each component that trigger notifications based upon Prometheus metrics. A custom Grafana dashboard was created that shows currently firing alerts.

v0.20.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: NOVEMBER 11, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 3, 2020  
{% endhint %}
```

This is a big release that contains support for a new HAPI service and whole new runtime component to dramatically improve performance. Due to the magnitude of the changes, it did take us a little longer to mark it as generally available as

we wanted to ensure it was tested as much as possible beforehand.

First up is support for the Hedera Token Service (HTS) that was recently announced and rolled out to previewnet. A lot of work was put into supporting the new transaction types on the parser side including enhancing the schema with new tables and ingesting them via the record stream. HTS also required a new balance file version that adds token information to the CSV. Token information is now returned for our existing REST APIs while the next release will contain token specific REST APIs for further granularity. Check it out in previewnet and let us know if you have any feedback!

We made a lot of strides in improving the ingestion performance in previous releases, but since we also wanted to ensure low end to end HCS latency via our gRPC API we had to sacrifice some of that speed. As a result, we could only ingest at about 3,000 transactions per second (TPS) before latency spiked above 10 seconds. This was entirely due to our use of PostgreSQL notify/listen to notify the gRPC API of new data.

In this release, we add a new notification mechanism without sacrificing speed or latency with our support for Redis pub/sub. With Redis, the mirror node can now ingest at least 10,000 TPS while still remaining under 10 seconds from submitting the topic message and receiving it back via the mirror node's streaming API. Redis is enabled by default, but it can be turned off if HCS latency is not a concern and you want to avoid another runtime dependency.

We also added support for the HAPI protobuf changes that are coming in v0.9.0. The protobuf is removing two deprecated fields while adding a new `signedTransactionBytes` field. Since the mirror node still needs to support historical transactions we retain support for parsing transactions that contain the old payload format.

v0.19.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: OCTOBER 6, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: SEPTEMBER 29, 2020  
{% endhint %}
```

This release finishes the State Proof alpha REST API and makes it generally available. As part of this, we made a lot of improvements to the check-state-proof command line tool that queries the API and validates the files locally. We also now store the node account used to verify record file, ensuring greater accuracy as to the provenance of the state proof.

There's been some changes to the public Hedera environments lately and we've updated the mirror node to reflect that. We added support for the new previewnet environment and we updated the configuration to point to the new testnet bucket after its recent reset. Please ensure your mirror node has all of the data in the previous bucket before updating to this release, assuming you're not specifying the bucket name manually.

We added proper liveness and readiness probe endpoints for all our components. If you're not familiar with the concept of liveness and readiness probes, check out the Kubernetes documentation on the subject. Our new liveness endpoint now does not fail if external dependencies are down like the database, ensuring the application doesn't restart unnecessarily. Even if you're not using Kubernetes it would be worthwhile to look into to ensure your mirror node is using the appropriate endpoint for health checks, based upon your needs.

v0.18.2

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: SEPTEMBER 22, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: SEPTEMBER 15, 2020  
{% endhint %}
```

Fix two regressions in the 0.18 release train.

v0.18.1

Contains a small change to the State Proof Alpha REST API to only return the current address book for now.

v0.18.0

Building upon the availability of the State Proof Alpha REST API in the last release, we've added sample code in JavaScript to retrieve the state proof from a mirror node and locally verify it. This allows users to obtain cryptographic proof that a particular transaction took place on Hedera. The validity of the proof can be checked independently to ensure that the supermajority of Hedera mainnet stake had reached consensus on that transaction. Similar to the promise of the ultimate state proofs, the user can trust this state proof alpha served by the mirror nodes, even when the user does not trust the mirror nodes.

Importer can now be configured to connect to Amazon S3 using temporary security credentials via AssumeRole. With this, a user that does not have permission to access an AWS resource can request a temporary role that will grant them that permission. See the configuration documentation for more information.

Importer also added two new properties to control the subset of data it should download and validate. The `hedera.mirror.importer.startDate` property can be used to exclude data from before this date and "fast-forward" to the point in time of interest. By default, the `startDate` will be set to the current time so mirror node operators can get up and running quicker with the latest data and reduce cloud storage retrieval costs. Note that this property only applies on the importer's first startup and can't be changed after that. The `hedera.mirror.importer.endDate` property can be used to exclude data after this date and halt the importer. By default it is set to a date far in the future so it will effectively never stop.

Breaking Changes

The aforementioned `startDate` property does change how the mirror node operators on initial start from previous releases. By defaulting to now, users standing up a new mirror node will no longer retrieve all historical data and will instead only retrieve the latest data. Current users upgrading to this release will not be affected even if their data ingest is not fully caught up since this property only applies if the database is empty like it is on first start. To revert to the previous behavior, a date in the past can be specified like the Unix epoch `1970-01-01T00:00:00Z`.

v0.17.3

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: SEPTEMBER 14, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: SEPTEMBER 3, 2020  
{% endhint %}
```

This release contains the port of a bug fix to better manage the `VertexException`:

Thread blocked issue seen in #945

v0.17.2

A small bug fix to better support resetting the mirror node when a stream reset is performed on the network environment

v0.17.1

A small fix to correct a performance regression with not properly caching a heavily used query.

v0.17.0

This release adds support for the storage of the network address books from file 0.0.101 and 0.0.102 in the mirror node database.\

The mirror node will now retrieve file address book contents which include node identifiers and their public keys from the database instead of the file system at startup.

This sets the stage for an additional feature which is the State Proof alpha REST API at /transactions/\${transactionId}/stateproof.\

With this release it is possible to request the address book, record file and signature files that contain the contents of a transaction and allow for cryptographic verification of the transaction. Mirror node users can now actively verify submitted transactions for themselves.

Other changes include support for continuous deployment (CD) using Github Actions that use FluxCD to deploy master versions to a Kubernetes cluster. Additionally, this release includes fixes to the database copy operation optimization and improved handling of buffer size used when copying large topic messages.

v0.16.0

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: AUGUST 18, 2020  
{% endhint %}
```

This release includes the foundation for some larger features to come. Notably, cloud bucket names are now set based on network specifications and users no longer need to explicitly state bucket names for demo, test and main networks. The recordfile table contents are also expanded to include the start and end consensus timestamps of their containing transactions. The recordfile table also saw a clean up to remove the path to the file.

Additionally, this release streamlines the helm chart architecture with a common chart for shared resources. It also adds dependabot to facilitate dependency update management. The parser was also updated to handle signature files across multiple time bucket groups for greater parsing robustness.

Memory improvements were also made in the parser to improve ingestion performance. Due to performance pg_notify was also removed in favor of direct psql_notify to support faster streaming of incoming topic messages.

v0.15.3

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

Works around an issue sending large JSON payloads via pg_notify by ignoring them for now. This occurs when a consensus message is sent with a message that exceeds 5824 bytes, which is also very close to the protobuf limit.

v0.15.2

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 20, 2020  
{% endhint %}
```

This release improves the topic message ingest rate that regressed in the previous release. This is just a stop gap and future releases will increase this further.

v0.15.1

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 15, 2020  
{% endhint %}
```

A hot fix release to address two high priority parsing errors with the new consensus message chunk header.

v0.15.0

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 15, 2020  
{% endhint %}
```

This release adds support for HCS topic fragmentation that will soon be rolled out to main nodes in the 0.6.0 release. For larger consensus messages that don't fit in the max transaction size of 6144 bytes, a standard chunk info header can be supplied to indicate how that message should be split into smaller messages. The Mirror Node now understands this chunk information and stores it in the database. Additionally, it will return this data when subscribing to the topic via the gRPC API. The Java SDK is being updated to automatically split and reconstruct this message as appropriate.

Other changes include optimizations around end to end latency of the gRPC API. This was accomplished mainly by adding a new `NotifyingTopicListener` that uses PostgreSQL's LISTEN/NOTIFY functionality.

v0.14.1

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 15, 2020  
{% endhint %}
```

This release further optimizes the ingestion rate. Initial tests indicate a 2x to 3x improvement.

v0.14.0

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 29, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 15, 2020  
{% endhint %}
```

This release is all about performance optimizations. We reworked some of the foreign keys to improve the ingestion performance by a few thousand transactions per second. We also fixed an out of memory issue with the gRPC API and did some optimizations in that area.

Besides performance, we made some other small improvements. We now set `topicRunningHashV2AddedTimestamp` with a default value for mainnet, making it not fail on startup if a value is not provided. Containerized acceptance and performance tests were added, making it easier to test at scale.

Breaking Changes

We removed `hedera.mirror.grpc.listener.bufferInitial` and `hedera.mirror.grpc.listener.bufferSize` properties since we removed the shared poller's buffer.

We also renamed some tables and columns which would affect you if you directly use the database structure. We renamed `ttransactions` to `transaction`, `tcryptotransferlists` to `cryptotransfer` and `nonfeetransfers` to `nonfeetransfer`.

v0.13.2

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 2, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JUNE 23, 2020  
{% endhint %}
```

Bug fix release to fix an out of memory issue with the gRPC API.

v0.13.1

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 2, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JUNE 23, 2020  
{% endhint %}
```

Small bug fix release to address gRPC NETTY issue blocking acceptance tests

v0.13.0

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JULY 2, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JUNE 23, 2020  
{% endhint %}
```

This release is a smaller release mainly focused on bug fixes with some minor enhancements. We added a new property `hedera.mirror.importer.downloader.endpointOverride` for testing. We also added `hedera.mirror.importer.downloader.gcpProjectId` to support specifying requester pays credentials with a personal account. Finally, we improved our Marketplace support getting us one closer to making it available.

v0.12.0

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: MAY 29, 2020  
{% endhint %}
```

This feature release contains a few nice additions while fixing a few critical bugs. We made good progress on adding our application to Google Cloud Platform Marketplace. This should be wrapping up soon and enable a "one click to deploy" of the mirror node to Google's Cloud. Additionally, some extra fields were added to our APIs. We added `runningHashVersion` to the REST and GRPC APIs. Finally, we added `transactionHash` to the transaction REST API.

We improved the importer ingestion rate from 3400 to 5600 transactions per second in our performance test environment. There's still room for improvement and we plan on making additional performance optimizations in an upcoming release.

Breaking Changes

We added an option to keep signature files after verification. By default, we no longer store signatures on the filesystem. If you'd like to restore the old behavior and keep the signatures, you can set `hedera.mirror.importer.downloader.record.keepSignatures=true` and `hedera.mirror.importer.downloader.balance.keepSignatures=true`.

We changed the bypass hash mismatch behavior in this release. Bypassing hash mismatch could be used in combination with other parameters to fast forward mirror node to newer data or to overcome stream resets. Previously you had to specify this via a database value in `tapapplicationstatus`. Since this data is not application state but considered more a user supplied value, we added a new property `hedera.mirror.importer.verifyHashAfter=2020-06-05T17:16:00.384877454Z` for this purpose.

v0.11.0

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JUNE 10, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: MAY 29, 2020  
{% endhint %}
```

This release was mainly focused on refactoring code and properties as a necessary step for future enhancements. We also continued making improvements to our Kubernetes support. To that end, we added Prometheus REST metrics, Helm tests and Mirror Node can now run in GKE.

We added a new parameter to all of the topic related REST APIs to return a topic message in plaintext instead of binary. Messages submitted to HAPI are submitted as binary and stored in the Mirror Node that way as well. If you know the messages are actually strings encoded in UTF-8, then you can set `encoding=utf-8` and the REST API will make a best effort conversion to string. By default or if you pass a query parameter of `encoding=base64`, it will return the message as base64 encoded binary.

Breaking Changes

Please note when upgrading that we made major breaking changes to the naming of our configuration properties. We've renamed all `hedera.mirror.api` properties to `hedera.mirror.rest`. We also renamed the properties `apiUsername` to `restUsername` and `apiPassword` to `restPassword` to reflect that as well. Any properties that were used by the importer module were renamed to be nested under `hedera.mirror.importer`. We apologize for any inconvenience.

We've removed the `hedera.mirror.addressBookPath` property in favor of a `hedera.mirror.importer.initialAddressBook` property. The former was overloaded to be both the initial bootstrap address book and the live address book being updated by file transactions for 0.0.102. The live address book is now hardcoded to `${hedera.mirror.importer.dataPath}/addressbook.bin` and cannot be changed.

The REST API to retrieve a topic message by its consensus timestamp now supports both a plural (`/topics/messages/:consensusTimestamp`) and singular (`/topic/message/:consensusTimestamp`) URI path. The singular format is deprecated and will be going away in the near future, so please update to the plural format soon.

We removed the singular form of a few alpha topic REST APIs. The `/topic/:id/message` API was removed in favor of the plural form `/topics/:id/messages`. Similarly, the `/topic/:id/message/:sequencenumber` API was removed in favor of its plural form `/topics/:id/messages/:sequencenumber`. Please update accordingly.

v0.10.1

Small bug fix release to address a REST API packaging issue.

v0.10.0

In preparation for Hedera Node release 0.5.0, we're releasing v0.10.0 to support the latest version of HAPI. The changes include renaming Claims to LiveHash and new response codes. One important HAPI change is the addition of a `topicRunningHashVersion` to the transaction record. This change was necessary as the way the topic running hash is changing with the release of 0.5.0. As a result, the Hedera Mirror Node added this new field to its database and a migration is ran to populate it with either the new or old version depending upon the release date of 0.5.0.

Unfortunately, this necessitated adding a required field `hedera.mirror.topicRunningHashV2AddedTimestamp` to control this behavior and will fail on startup if this is not populated. This is just a temporary measure. Once Hedera Node 0.5.0 is released to testnet and mainnet we will update this so it's automatically populated with the known date.

Other changes include adding Google PubSub support to publish JSON representing the Transaction and TransactionRecord protobuf to a message queue for external consumption. We've also added REST API metrics and added Traefik as an API gateway for our helm chart.

Breaking changes

We've had to remove our event stream support. This area of the code was never enabled and was untested and was incurring technical debt without providing any benefit. If it becomes necessary in the future, we can re-add it within our newly refactored framework.

The new `/api/v1/topics/:id` alpha REST API that was added in 0.9 has been changed to `/api/v1/topics/:id/messages`. This change was made to align the API with the other topic message APIs as it refers to the messages entity and not the topic entity.

v0.9.1

Small bug fix release to address not being able to handle address book updates that span multiple transactions.

v0.9.0

This release contains another new REST API for our consensus service. You can now retrieve all topic messages in a particular topic, with additional filtering by sequence number and consensus timestamp. Here's an example:

```
GET /api/v1/topic/7?
```

```
sequencenumber=gt:2&timestamp=lte:1234567890.000000006&limit=2
```

```
{
  "messages": [
    {
      "consensustimestamp": "1234567890.000000003",
      "topicid": "0.0.7",
      "message": "bWVzc2FnZQ==",
      "runninghash": "cnVubmluZ19oYXNo",
      "sequencenumber": 3
    },
    {
      "consensustimestamp": "1234567890.000000004",
      "topicid": "0.0.7",
      "message": "bWVzc2FnZQ==",
      "runninghash": "cnVubmluZ19oYXNo",
      "sequencenumber": 4
    }
  ],
  "links": {
    "next": "/api/v1/topic/7?
sequencenumber=gt:2&timestamp=lte:1234567890.000000006&timestamp=gt:1234567890.0
00000004&limit=2"
  }
}
```

The other big feature of this release is Kubernetes support. We've create a Helm chart that can be used to deploy a highly available Mirror Node with a single command. This feature is still under heavy development as we work towards converting our current deployments to this new approach.

v0.8.1

Small bug fix release to fix a packaging issue.

v0.8.0

Mirror node v0.8.0 is here! We're made great strides in making the mirror node easier to run and manage. In particular, we added support for requester pays buckets. This will allow anyone to run a mirror node as long as they are willing to pay for the cost to retrieve the data. Currently only Hedera and a few partners have access to the bucket, so enabling this will open up that data to our community. We are still working on a migration of the buckets to this model, so stay tuned.

We also added two new experimental REST APIs to retrieve HCS data. Firstly, we added `/api/v1/topic/message/${consensusTimestamp}` to retrieve a topic message by its consensus timestamp. Secondly, we added `/api/v1/topic/${topic}/message/${seqNum}` to retrieve a particular topic message by its sequence number from a

topic. These APIs are considered alpha and may be changed or removed in the future. We also dramatically increased test coverage for the REST APIs and squashed some bugs in the process.

For our GRPC API, we had to switch from R2DBC to Hibernate to reach the scale and stability that we needed. In doing so, we can now support a lot more concurrent subscribers at a higher throughput. It should also finally put to rest any stability concerns with the GRPC component.

There are a few breaking changes that we had to make. We now no longer write and store record or balance files to the filesystem after they are inserted into the database. If you still need these files, you can set `hedera.mirror.parser.balance.keepFiles` and `hedera.mirror.parser.record.keepFiles` to `true`.

Also, we moved the persist properties to be grouped under a new path. That is we moved options like `hedera.mirror.parser.record.persistTransactionBytes` to `hedera.mirror.parser.record.persist.transactionBytes`. Please update your local configuration accordingly.

v0.7.0

0.7.0 focuses on refactoring the record file parsing to decouple the parsing from the persisting of data. This refactoring is laying the groundwork for additional performance improvements and allowing additional downstream components to register for notification of the transactions.

v0.6.0

Release focused on stability and performance improvements.
End to end test coverage.

This release was mainly focused on enhancing the stability and performance of the mirror node. We improved the transaction ingestion speed from 600 to about 4000 transactions per second. At the same time, we greatly improved the resiliency and performance of the GRPC module. We also added acceptance tests to test out HCS end to end.

Breaking Change

Please note that one potentially breaking change in this release is to reject subscriptions to topics that don't exist. This avoids the server having to poll repeatedly until it is created and taking up resources for a topic that may never exist. It is expected that clients or the SDK will poll periodically after creating a topic until that topic makes its way to the mirror node. This functionality is hidden behind a feature flag but will slowly be rolled out over the next month.

v0.5.3

Now supports all HCS functionality including a streaming gRPC API for message topic subscription.

Changed how the mirror node verifies mainnet consensus. Mirror node now waits for at least third of node signatures rather than greater than two thirds to verify consensus.

Added new mainnet nodes to the mirror node address book.

Access still restricted to a white listed set of IP addresses. Request access [here](#).

Please see the Mirror Node releases page for the full list of changes here.

We occasionally may encounter a situation where an additional 15-20 second delay in message round trip time is experienced and subscriber connections are dropped. No messages are lost, and the consensus time is not affected. Clients are encouraged to reconnect. This issue will be fixed in a subsequent release of the Hedera mirror node. Some third-party mirror nodes should not be affected by

this issue. We also don't expect it to impact the exchanges using the REST end point for the mirror node.

README.md:

Release Notes

services.md:

description: Hedera Services release information

Hedera Services

Please visit the Hedera status page for the latest versions supported on each network.

Release 0.53

{% hint style="info" %}
MAINNET UPDATE SCHEDULED: SEPTEMBER 11, 2024
{% endhint %}

{% hint style="info" %}
TESTNET UPDATE SCHEDULED: SEPTEMBER 4, 2024
{% endhint %}

Build 0.53.5

What's Changed

- feat: add enabledDAB flag to enable and disable DAB features by @iwsimon in #15232
- ci: resolves release issue preventing the publication of the docker images by @nathanklick in #15158
- fix: hedera-evm and hedera-evm-impl are overwriting each other in MC by @rbarkerSL in #15175

Build 0.53.1

What's changed

- fix: change order of descriptor variables by @lpetrovic05 in #15016

Build 0.53.0

What's changes

- docs: 13690 Added a design doc for Ledger State API by @imalygin in #13730
- chore: update Gradle to 8.8 / setup-gradle to v3.4.2 by @jjohannes in #13757
- chore: Cleanup obsolete test-clients code and resources by @tinker-michaelj in #14050
- docs: update token reject design doc by @MiroslavGatsanoga in #14061
- fix: passing upgrade @HapiTest by @tinker-michaelj in #13992
- feat: Ensure overwritten operations check for sufficient gas first by @lukelee-sl in #11441
- test: HIP-904 Create HAPI tests for a hollow account on an alias on which we have a deleted account by @zhpetkov in #14036
- feat: HIP-904 Charge automatic associations during CryptoTransfer by @Neeharika-Sompalli in #14107

chore: inline pces proposal 2.0 by @cody-littlely in #14056
feat: implement HIP-632 isAuthorizedRaw method by @david-bakin-sl in #14130

➡ See the full list of changes here.\

Release 0.52

Release Highlights

HIP-632

isAuthorizedRaw Functionality

Accepts three parameters:

- address
- messageHash
- signatureBlob

Validates the provided address (Hedera Account ID or virtual address)

Determines signature type based on signatureBlob length:

- 65 bytes: ECDSA
- 64 bytes: ED25519

ECDSA Signature Handling

Extracts v, r, and s components

Runs ECRECOVER to recover signing address

Compares result with the account's virtual addresses

ED25519 Signature Handling

Retrieves Hedera address

Checks for single associated key on account

Verifies signature against message hash and account key

Benefits

Similar functionality to Ethereum's ECRECOVER precompile

Supports both ECDSA and ED25519 signature verification

Works with Hedera Account IDs and virtual addresses

Simplifies signature verification in smart contracts

Streamlines transaction authentication within contracts

Enhances Hedera-Ethereum authorization flow compatibility

Improves developer experience with familiar authorization mechanism

HIP 904

TokenReject Functionality

Allows users to reject undesired tokens

Transfers the full balance of one or more tokens from the requesting account to the treasury

Supports rejection of both fungible and non-fungible tokens

Handles up to 10 token rejections in a single transaction

Bypasses custom fees and royalties defined for the rejected tokens

Benefits of TokenReject

Enables users to remove unwanted tokens from their accounts

Protects users from potential scams or unwanted airdrops

Allows rejection of tokens regardless of how they were acquired (manual or automatic association)

Helps users manage their token holdings more effectively

Prevents users from being forced to pay exorbitant or potentially malicious

fees to remove tokens

Maintains account association with the token, allowing for future transactions if desired

Provides a simple mechanism for users to clean up their accounts

Enhances user control over their token portfolio

Improves overall user experience in token management

Build 0.52.3

{% hint style="success" %}

MAINNET UPDATE SCHEDULED: AUGUST 28, 2024

{% endhint %}

What's Changed

fix: invalid feeSchedules.json by @Neeharika-Sompalli in #14881

Build 0.52.2

{% hint style="success" %}

TESTNET UPDATE SCHEDULED: AUGUST 14, 2024

{% endhint %}

What's Changed

chore(0.52): updates the buildkit and docker daemon configuration to use the registry mirror by @nathanklick in #14777

fix: immediately finalize transfer lists for scheduled crypto transfer by @tinker-michaelj in #14799

Build 0.52.1

What's Changed

ci: fix gradle publish failures in release-0.52 for hedera.com.evm by @rbarkerSL in #14513

fix: 14489 cherry pick docker rate limit fix in release052 by @rbarkerSL in #14490

fix(bug): Removed daemon config changes (#14599) by @rbarkerSL in #14602

fix: cherry pick misc fixes by @tinker-michaelj in #14609

Build 0.52.0

{% hint style="success" %}

TESTNET UPDATE SCHEDULED: JULY 31, 2024

{% endhint %}

What's Changed

The accounts.maxNumber and nfts.MaxAllowedMints values both remain at 20 million for this release

feat: extract HederaNetwork interface with initial SubProcessNetwork impl by @tinker-michaelj in #13540

build: make annotation library dependencies transitive by @jjohannes in #13643

chore: Address compiler warnings in LoggerApiSpecAssertions by @jjohannes in #13644

chore: disabled new backpressure via settings by @cody-littlely in #13635

chore: Add FakePlatform and FakeServicesRegistry by @Neeharika-Sompalli in #13549

docs: File Service design doc by @derektriley in #13615

build: (de)activate selection of javac lint features by @jjohannes in #11838

fix(reconnect): use AtomicLong for anticipatedMessages counter by @anthony-swirlslabs in #13650

feat: Move to fully connected network by @kfa-aguda in #13010

docs: add design document for HIP-904 token cancel airdrop transaction by @MiroslavGatsanoga in #12787

➞ See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.51

```
{% hint style="success" %}  
MAINNET UPDATE: JULY 17, 2024  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: JULY 2, 2024  
{% endhint %}
```

Release Highlights

HIP 206

Functionality

Defines a new function to the Hedera Token Service system contract that allows for the atomic transfer of HBAR, fungible tokens and non-fungible tokens.

Function `cryptoTransfer(TransferList transferList,TokenTransferList\[] tokenTransfer)`

Exposes an existing HAPI call via smart contracts.

Transfer respects granted allowances.

Benefits

Enables native royalty support on the EVM since native \$hbar can now be transferred using spending allowances

Direct interaction with HBAR and HTS tokens

Eliminates the need for token wrapping.

Enhances efficiency and reduces complexity.

Cuts costs by removing intermediary steps i.e., wrapping assets to interact with them.

Enables native royalty support on the EVM since native HBAR can now be transferred using spending allowances

HIP 906

Functionality

Introduces a new Hedera Account Service system contract.

Enables querying and granting approval of HBAR to a spender account from smart contracts code

`hbarAllowance`, `hbarApprove`

Developers do not have to context switch out of smart contract code

Benefits

Introduces new account proxy contract for HBAR allowances

Enables grant, retrieve, and manage HBAR allowances within smart contracts

Developers do not have to context switch out of smart contracts code

Simplifies workflows and enhances security

Expands potential use cases, especially for DeFi and token marketplaces

0.51.5

What's Changed

feat(reconnect): introduce ReconnectMapStats interface by @anthony-swirldslabs in #13027
chore: revert removal of CLI report tool by @lpetrovic05 in #13002
docs: add design document for HIP-904 token reject operation by @MiroslavGatsanoga in #12786
feat: gossip facade by @cody-littlely in #12897
feat: add the ability to disable the running event hasher by @cody-littlely in #13083
fix: ignore token expiry status in TokenDissociate by @tinker-michaelj in #13104
feat: add javadoc and diagram, delete dead code by @tinker-michaelj in #13070
fix: use civilian payer for modified variants by @tinker-michaelj in #13020
fix: 12853: Memory leak from MerkleDbDataSource.copyStatisticsFrom() by @artemananiev in #13097
feat: Updated hedera-services code to support DAB protobuf changes. by @iwsimon in #13090

[→](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.50

```
{% hint style="success" %}  
MAINNET UPDATE: JUNE 20, 2024  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: JUNE 5, 2024  
{% endhint %}
```

0.50.1

What's Changed

chore: Cherry pick 13648 into release 0.50 branch by @lukelee-sl in #13662
fix(ci): cherry pick milestone assignee checks rel 50 by @rbarkerSL in #13712
fix: (cherry-pick) Use restart method to all token schemas by @Neeharika-Sompalli in #13676
fix: Enable tokens.balancesInQueries.enabled by @netopyr in #13716
chore: Enable tokens.balancesInQueries in code by @netopyr in #13769

[→](#) See the full list of changes here.

0.50.0

What's Changed

feat: reorganize ISS wiring by @alittlely in #11685
feat(diff-testing): Script (python) to pull intervals - up to a day - from GCP by @david-bakin-sl in #11409
fix: 11750 Fixed synchronization in BreakableDataSource.saveRecords by @imalygin in #11756
feat: Differential testing: Enhance account store dumper to handle modular representation by @vtronkov in #11489
test: add security v2 model tests for token associate by @anastasiya-kovaliova in #11327
fix: stop checking for minimum birth round by @cody-littlely in #11769

feat: make the state compatible with birth rounds by @cody-little in #11780
fix: FilteredLoggingMonitor by @mxtartaglia-sl in #11754
feat: diagram tweaks by @cody-little in #11801
fix: wait longer for freeze transaction to be handled by @JeffreyDallas in #11790

[↩](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.49

{% hint style="success" %}
MAINNET UPDATE: MAY 22, 2024
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: MAY 14, 2024
{% endhint %}

0.49.7

What's Changed

fix: support crypto admin keys in system contract tokenCreate() by @tinker-michaelj in #13148
fix: remove balance adjustment limit from record in state, use 0 for initial gas snapshot by @tinker-michaelj in #13185

0.49.6

What's Changed

fix: cherry-pick midnight rate management on restart (#13071) by @povolev15 in #13091
feat: auto-resubmit operations with modifications (#12811) by @Neeharika-Sompalli in #13088
fix: ignore token expiry status in TokenDissociate by @tinker-michaelj in #13106
fix: avoid NPE when migrating from genesis (non-prod) state by @tinker-michaelj in #13123

0.49.5

What's Changed

fix: storage link management by @tinker-michaelj in #13056

0.49.1

What's Changed

fix: manage StakingInfos in restart by @tinker-michaelj in #12911

0.49.0

What's changed

feat: address cold read issue in ExtCodeHash operation by @lukelee-sl in #11323
fix: 11348: The fix for 11231 doesn't cover ParsedBucket by @artemananiev in #11349

chore: Create ISS detector component by @lpetrovic05 in #11075
chore: Add orderedSolderTo method to OutputWire by @poulok in #11330
chore: remove hashgraph demo by @lpetrovic05 in #11352

[→](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.48

{% hint style="success" %}
MAINNET UPDATE: APRIL 25, 2024
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 18, 2024
{% endhint %}

0.48.1

What's Changed

fix: remove adjustments limit by @tinker-michaelj in #12826

0.48.0

{% hint style="success" %}
TESTNET UPDATE: APRIL 11, 2024
{% endhint %}

What's Changed

feat: Check platform status before syncing (#11429) by @alittlely in #12679

Performance Results

<figure><figcaption></figcaption></figure>

v0.47

{% hint style="success" %}
MAINNET UPDATE: APRIL 4, 2024
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: MARCH 28, 2024
{% endhint %}

0.47.4

What's Changed

chore: cherry-pick unified CryptoCreate throttle reclamation (#12339).

0.47.3

{% hint style="success" %}
TESTNET UPDATE: MARCH 20, 2024
{% endhint %}

What's Changed

chore: Configure maxAggregateRels to 15 million (all envs) (#12053).

0.47.2

What's Changed

fix: Update Configuration hashesRamToDiskThreshold to 0 in MerkleDbConfig
fix: Backport the fix for virtual map flushes.

0.47.1

{% hint style="success" %}
TESTNET UPDATE: FEBRUARY 29, 2024
{% endhint %}

What's Changed

fix: only compare child time created against self-parent time created by @alittle in #11673
chore: add an old-style queue thread for intake by @cody-little in #11671
fix: 11746: Backport the fix for #11304 to release 0.47 by @artemmananiev in #11747

0.47.0

What's Changed

fix: bug when node is removed by @cody-little in #10687
fix: Fuzzy matching for CreateOperationSuite and Create20perationSuite 09431 by @JivkoKelchev in #10185
fix: recordCache to commit added entries and implemented correctly the remove elements from the queue by @povolev15 in #10523
fix: Fix and enable all Schedule HapiTests by @povolev15 in #10551
fix: implement sidecars by @JivkoKelchev in #9815
feat: add setting for birth round ancient threshold by @cody-little in #10660
chore: drop chatter by @cody-little in #10670
chore: remove state info by @cody-little in #10685
chore: Rename contract causing services regression due to long name by @stoqnkpl in #10700
fix: state leak by @cody-little in #10690

[↩](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.46

{% hint style="success" %}
MAINNET UPDATE: FEBRUARY 21, 2024
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: FEBRUARY 6, 2024
{% endhint %}

0.46.3

What's Changed

chore: bump HAPI proto version by @tinker-michaelj in #11232

0.46.2

```
{% hint style="success" %}  
TESTNET UPDATE: JANUARY 30, 2024  
{% endhint %}
```

What's Changed

fix: Ensure that the pending creation customizer applies to the address being created by @lukelee-sl in #11213

0.46.1

What's Changed

chore: bump HAPI proto version by @tinker-michaelj in #11232

0.46.0

```
{% hint style="success" %}  
TESTNET UPDATE: JANUARY 23, 2024  
{% endhint %}
```

What's Changed

feat: wiring diagram improvements by @cody-littlej in #10233
chore: Change HashMap to LinkedHashMap in custom fees assessment by @Neeharika-Sompalli in #10240
feat: add implementation in throttling facility to handle N-Of-Unscaled type of throttling by @MiroslavGatsanoga in #10142
build: do not publish test fixtures by @jjohannes in #10147
build: patch everything we use to be a real Java Module by @jjohannes in #10056
chore!: More common tests moved to correct module by @hendrikebbers in #10133
feat: Config constants created & used by @hendrikebbers in #10117
feat: script for cleaning build files by @cody-littlej in #10190
fix: Compact last PCES file at boot time by @cody-littlej in #10257
feat: sync+- by @cody-littlej in #10260

[→](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.45

```
{% hint style="success" %}  
MAINNET UPDATE: JANUARY 9, 2024  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: DECEMBER 28, 2023  
{% endhint %}
```

0.45.2

What's Changed

fix: Added a feature flag which is by default enabled to disable tokenBalances and tokenRelationships in getAccountInfo, getAccountBalance and getContractInfo queries. #10639

0.45.0

Populate evm function result on failing eth transaction by @stoqnkpL in #9453
Disable compression. by @cody-little in #9554
Fix tests in unique token management spec by @mhess-swl in #9537
enabled one more test and remove the other one that not really in use by @povolev15 in #9557
Enable tests from CannotDeleteSystemEntitiesSuite by @Ivo-Yankov in #9440
Fix tests in ContractBurnHTSSuite by @agadzhalov in #9572
Tune dependency scopes by @jjohannes in #8455
unneeded calls to swirlds-common removed by @hendrikebbers in #9003
Fixed CryptoRecordsSanityCheckSuite by @iwsimon in #9551
Enable test from AssociatePrecompileSuite by @mustafauzun in #9571

Performance Results

<figure><figcaption></figcaption></figure>

v0.44

{% hint style="success" %}
MAINNET UPDATE: DECEMBER 19, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: DECEMBER 12, 2023
{% endhint %}

0.44.3

What's Changed

Enforce NFT allowance check on auto-creation by @tinker-michaelj in e69d0a9

0.44.2

What's Changed

Catch UncheckedIOException during PCES file copy. (#10083) by @cody-little in #10087

0.44.1

Bug Fixes

Fix PCES copy bugs. (#10057) #10062

0.44.0

Features

Re-add bootstrap.properties file to maintain downstream processes and increase accounts.maxNumber=20\000\000 #8915
8815: sort dirty leaves during flush #8981
Add setting to disable critical quorum. #8961
Add a doc for all system entity numbers #8993
08566 - Validate PCES Events When Loading State On Different Network #8568
Differential testing analytic engine: State file file dumper now dumps special files #8991
Added improved startup ASCII art. #9028
Characterize invalid id failure modes for classic HTS calls #9053
Add ordinals to status diagram, and update javadocs #9108

5552: Create a Grafana Data Dashboard to view all existing relevant data metrics #8845
Update Besu to version 23.10.0 #9168

[→](#) See the full list of changes here.

Performance Results

<figure><figcaption></figcaption></figure>

v0.43

{% hint style="success" %}
MAINNET UPDATE: NOVEMBER 27, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: NOVEMBER 2, 2023
{% endhint %}

Services v0.43.0 adds the following features:

HIP-786 (#8620)

Enhancements

Services v0.43.0 adds the following enhancements:

Update Besu to 23.10.0 - cherry pick (#9199)
Update the Besu EVM library to version 23.7.2 (#8472)
"Productizing" contract disassembler at last (#8563)
Auto sidecar validations (#8404)
Create fat jar with services CLI so it can be run standalone (#8519)

Performance Results

<figure><figcaption></figcaption></figure>

v0.42

{% hint style="success" %}
MAINNET UPDATE: OCTOBER 24, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: SEPTEMBER 26, 2023
{% endhint %}

0.42.6

This release updates the platform SDK version from 0.42.0 to 0.42.6, which removes reconnect.asyncStreamTimeout from the settings files. Doing so ensures that this property will default to the value specified in code (300 seconds).

Changes

Upgrade platform SDK (#9224)

0.42.2

Changes

0.42 account balance test (#8866)
Re-add bootstrap.properties file and increase accounts.maxNumber=20000000
(#8928)

0.42.1

Changes

Chore: normalize configuration values (release/0.42) (#8668)
8751: No data source metrics for accounts, NFTs, or token rels (#8798)

0.42.0

Add EIP 2930 support to EthTXData (#7696)
Provide entity and throttle dashboards (#7774)
07748 Postconsensus signature gathering (#7776)
Enable EIP-2930 transactions by default (#7786)
7570: Remove JasperDB (#7803)
Remove support for legacy sync gossip. (#8059)
Disable account balance exports (#8272)
Modify config to support state on disk by default (#8510)

Performance Results

<figure><figcaption></figcaption></figure>

v0.41

{% hint style="success" %}
MAINNET UPDATE: SEPTEMBER 20, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: AUGUST 22, 2023
{% endhint %}

Ethereum transaction type support is expanded to include type 1 transactions (#7670) which follow EIP 2930 RLP encoding. This increases the number of native EVM tools and scenarios the Hedera Smart Contract Service supports.

NFT mint pricing is changed to linearly scale based on number of serials minted. Also, minting a single NFT in collection is changed to cost \$0.02 from \$0.05. #7769

Performance Results

<figure><figcaption></figcaption></figure>

v0.40

{% hint style="success" %}
MAINNET UPDATE: AUGUST 15, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: AUGUST 8, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: JULY 19, 2023
{% endhint %}

The 0.40 release of Hedera Services delivers HIP-729 \ "Contract Accounts Nonce

Externalization". Smart contract developers using the Hedera public mirror node can now track contract nonces as they would on e.g., Ethereum. Use cases might include troubleshooting failed contract calls or writing unit tests that validate transaction ordering based on CREATE1 addresses (once these are set by default in release 0.41+).

Open source contributors to the project will notice major refinements in the Gradle build, thanks to @jjohannes's expert touch.

Performance Results

<figure><figcaption></figcaption></figure>

v0.39

```
{% hint style="success" %}
MAINNET UPDATE: JULY 11, 2023
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: JUNE 21, 2023
{% endhint %}
```

Services v0.39.0 adds the following features:

VirtualRootNode constructor creates a cache object that doesn't get reused #6321

Implement blocklisting of EVM addresses #5799

Optimize virtual node cache flush strategy #5568

HIP-721: 06026 - add software version to events #6236

Implement CryptoCreate handle method #6112

UtilPrng handle Implementation #6310

Add a PCLI sub command to sign services stream files #6309

Implement token freeze handling #6467

Implement token unfreeze handle() #6502

Combine Admin and Network modules #6511

Implement the modular Pre-Handle Workflow #6291

Move hashes out of leaves node in VirtualMap #5825

TokenFeeScheduleUpdate handle() implementation #6582

Basic File Service implementation #6522

Implement Token Association to Account #6609

Implementation of handle workflow #6476

Implement the modular record cache #6754

CryptoDelete handle implementation #6694

Performance Results

<figure><figcaption></figcaption></figure>

v0.38

```
{% hint style="success" %}
MAINNET UPDATE: JUNE 8, 2023
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: JUNE 1, 2023
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: MAY 18, 2023
{% endhint %}
```

Upgrade EVM to Shanghai #5964
EVM version update and optimizations #5962
Turn on the Shanghai version of the EVM in previewnet #6212
Update hedera-protobufs-java version to 0.38.10 #6579
Add PCLI command to sign account balance files #6264

Performance Results

<figure><figcaption></figcaption></figure>

v0.37

{% hint style="success" %}
MAINNET UPDATE: MAY 17, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 24, 2023
{% endhint %}

Features

Implement topic deletion prehandle (#5033)
Generalize workflows enabled and add workflow ports (#5032)
Pre-handle improvements (#5056)
Support auto-scheduling operations by type within a suite (#5054)
Add SPI and App components supporting TransactionDispatcher for modularized HCS (#5062)
added the missing functionality to FileSignTool (#5100)
Consensus Message Submission Prehandle (#5059)
Add IngestChecker mono adapters for sigs and solvency (#5098)
\[HIP-583] Finalize hollow accounts via any required signature in a txn (#4990)
Remove CryptoCreate capability to create hollow accounts (#4998)
Populate EVM Address in CryptoTranscation (#5010)
Enable All EVM E2E suites to run with Ethereum Calls (#4375)

Performance Results

<figure><figcaption></figcaption></figure>

v0.36

{% hint style="success" %}
MAINNET UPDATE: APRIL 20, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 13, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 4, 2023
{% endhint %}

Features

Services v0.36.0 adds the following functionality:

Add tracking of property changes for hollow account completion (#4647)
Adding support for Redirect Token Calls fro evm-module (#4880)
Update FileSignTool (#4988)

Adding block number tool (#4997)
Add client.workflow.operations and test with workflows (#5053)
update hedera-services to use FSTS CLI instead of system properties
6166: Migrate VirtualMap data from JasperDB to MerkleDb data sources
Implementation of current network functionality in new, modularized application architecture: consensus operations, query workflow, and various preHandle implementations

Security Updates: Hedera Smart Contract Service Security Model Changes

Changes from services v0.35.2 have also been ported to v0.36.0.

After the security incident on March 9th, the engineers conducted a thorough analysis of the Smart Contract Service and the Hedera Token Service system contracts.

As part of this exercise, we did not find any additional vulnerabilities that could result in an attack that that which we witnessed on March 9th.

The team also looked for any disparities between the expectations of a typical smart contract developer who is used to working with the Ethereum Virtual Machine (EVM) or ERC token APIs and the behaviors of the Hedera Token Service system contract APIs. Such differences in behavior could be used by a malicious smart contract developer in unexpected ways.

In order to eliminate the possibility of these behavioral differences being utilized as attack vectors in the future, the consensus node software will align the behaviors of the Hedera Smart Contract Service token system contracts with those of EVM and typical token APIs such as ERC 20 and ERC 721.

As a result, the following changes are made as of the mainnet 0.35.2 release on March 31st:

An EOA (externally owned account) will have to provide explicit approval/allowance to a contract if they want the contract to transfer value from their account balance.

The behavior of transferFrom system contract will be exactly the same as that of the ERC 20 and ERC 721 spec transferFrom function.

For HTS specific token functionality (e.g. Pause, Freeze, or Grant KYC), a contract will be authorized to perform the associated token management function only if the ContractId is listed as a key on the token (i.e. Pause Key, Freeze Key, KYC Key respectively).

The transferToken and transferNFT APIs will behave as transfer in ERC20/721 if the caller owns the value being transferred, otherwise it will rely on approve spender allowances from the token owner.

The above model will dictate entity (EOA and contracts) permissions during contract executions when modifying state. Contracts will no longer rely on Hedera transaction signature presence, but will instead be in accordance with EVM, ERC and ContractId key models noted.

As part of this release, the network will include logic to grandfather in previous contracts.

Any contracts created from this release onwards will utilize the stricter security model and as such will not have considerations for top-level signatures on transactions to provide permissions.

Existing contracts deployed prior to this upgrade will be automatically grandfathered in and continue to use the old model that was in place prior to this release for a limited time to allow for DApp/UX modification to work with the new security model.

The grandfather logic will be maintained for an approximate period of 3 months from this release. In a future release in July 2023, the network will remove the grandfather logic, and all contracts will follow the new security model.

Developers are encouraged to test their DApps with new contracts and UX using the new security model to avoid unintended consequences. If any DApp developers fail to modify their applications or upgrade their contracts (as applicable) to adhere to the new security model, they may experience issues in their applications.

Performance Results

<figure><figcaption></figcaption></figure>

v0.35

```
{% hint style="success" %}  
MAINNET UPDATE: MARCH 31, 2023  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: MARCH 16, 2023  
{% endhint %}
```

0.35.2 Hedera Smart Contract Service Security Model Changes

After the security incident on March 9th, the engineers conducted a thorough analysis of the Smart Contract Service and the Hedera Token Service system contracts.

As part of this exercise, we did not find any additional vulnerabilities that could result in an attack that that which we witnessed on March 9th.

The team also looked for any disparities between the expectations of a typical smart contract developer who is used to working with the Ethereum Virtual Machine (EVM) or ERC token APIs and the behaviors of the Hedera Token Service system contract APIs. Such differences in behavior could be used by a malicious smart contract developer in unexpected ways.

In order to eliminate the possibility of these behavioral differences being utilized as attack vectors in the future, the consensus node software will align the behaviors of the Hedera Smart Contract Service token system contracts with those of EVM and typical token APIs such as ERC 20 and ERC 721.

As a result, the following changes are made as of the mainnet 0.35.2 release on March 31st:

An EOA (externally owned account) will have to provide explicit approval/allowance to a contract if they want the contract to transfer value from their account balance.

The behavior of transferFrom system contract will be exactly the same as that of the ERC 20 and ERC 721 spec transferFrom function.

For HTS specific token functionality (e.g. Pause, Freeze, or Grant KYC), a contract will be authorized to perform the associated token management function only if the ContractId is listed as a key on the token (i.e. Pause Key, Freeze Key, KYC Key respectively).

The transferToken and transferNFT APIs will behave as transfer in ERC20/721 if the caller owns the value being transferred, otherwise it will rely on approve spender allowances from the token owner.

The above model will dictate entity (EOA and contracts) permissions during contract executions when modifying state. Contracts will no longer rely on Hedera transaction signature presence, but will instead be in accordance with EVM, ERC and ContractId key models noted.

As part of this release, the network will include logic to grandfather in previous contracts.

Any contracts created from this release onwards will utilize the stricter security model and as such will not have considerations for top-level signatures on transactions to provide permissions.

Existing contracts deployed prior to this upgrade will be automatically grandfathered in and continue to use the old model that was in place prior to this release for a limited time to allow for DApp/UX modification to work with the new security model.

The grandfather logic will be maintained for an approximate period of 3 months from this release. In a future release in July 2023, the network will remove the grandfather logic, and all contracts will follow the new security model.

Developers are encouraged to test their DApps with new contracts and UX using the new security model to avoid unintended consequences. If any DApp developers fail to modify their applications or upgrade their contracts (as applicable) to

adhere to the new security model, they may experience issues in their applications.

Features

HIP-583 to expand alias support in CryptoCreate & CryptoTransfer Transactions.

This includes,

CryptoTransfer to non-existing EVM address alias causing hollow-account creation.

Finalizing a hollow account with the payer signature in an incoming transaction

Use cases for HIP-583 that work in this release :

1. As a user with an ECDSA based account from another chain I can have a new Hedera account created based on my evm-address alias.
2. As a developer, I can create a new account using a evm-address alias via the CryptoTransfer transaction.
3. As a developer, I can transfer HBAR or tokens to a Hedera account using their evm-address alias.
4. As a Hedera user with an Ethereum-native wallet, I can receive HBAR or tokens in my account by sharing only my evm-address alias.
5. As a Hedera user with a Hedera-native wallet, I can transfer HBAR or tokens to another account using only the recipient's evm-address alias.

Configuration Changes

```
autoCreation.enabled=true
lazyCreation.enabled=true
cryptoCreateWithAliasAndEvmAddress.enabled=false
contracts.evm.version=v0.34
```

Performance Results

<figure><figcaption></figcaption></figure>

v0.34

```
{% hint style="success" %}
MAINNET UPDATE: FEBRUARY 9, 2023
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: JANUARY 24, 2023
{% endhint %}
```

0.34.3

Use v0.34.3 SDK.

0.34.0

Services v0.34.0 completes the implementation of HIP-583.

To ensure full test coverage of this intricate feature, it will first be enabled only on previewnet.

This release will not enable smart contract rent.

Performance Results

<figure><figcaption></figcaption></figure>

v0.33

{% hint style="success" %}
MAINNET UPDATE: JANUARY 12, 2023
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: DECEMBER 22, 2022
{% endhint %}

Services v0.33.0 adds the following features:

- Hyperledger Besu EVM updated to version 22.10.x
- 'accounts send' subcommand added to yahcli to support sending HTS token units
- Developer documentation updates

<figure><figcaption></figcaption></figure>

v0.31

{% hint style="success" %}
MAINNET UPDATE: DECEMBER 9, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: DECEMBER 1, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: NOVEMBER 11, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: OCTOBER 27, 2022
{% endhint %}

Services 0.31 completes the following features:

- HIP-542 roadmap for making payer of the CryptoTransfer sponsor for auto-creation. It also enables auto-creation with Token transfers in addition to Hbar transfers.

- HIP-564 roadmap for allowing zero unit fungible token transfers

- HIP-573 roadmap for enabling token creators an option to exempt all of their token's fee collectors from a custom fee.

In addition to the above features,

- Adds support of the ERC20/721 transferFrom method for HTS precompiles from HIP-514 roadmap.

- Enables Smart Contract Traceability.

- Adds some changes related to testability improvements.

<figure><figcaption></figcaption></figure>

v0.30

{% hint style="success" %}
MAINNET UPDATE: OCTOBER 21, 2022

{% endhint %}

{% hint style="success" %}

TESTNET UPDATE: OCTOBER 19, 2022

{% endhint %}

{% hint style="success" %}

TESTNET UPDATE: OCTOBER 6, 2022

{% endhint %}

Services 0.30 completes the HIP-514 roadmap for making Hedera native tokens manageable via smart contracts. There are five new system contracts: `getTokenExpiryInfo(address)`, `updateTokenExpiryInfo(address, Expiry)`, `isToken(address token)`, `getTokenType(address token)`, and `updateTokenInfo(address, HederaToken)`.

The `updateTokenInfo(address, HederaToken)` call is especially powerful. If a token's admin key signs the transaction calling a contract, that contract can now make itself the token's treasury, assume authority to mint or burn units or NFTs, and so on.

⚠ Contract authors should know this release initiates Hedera's expiration and rent model for contracts. There will be two visible effects immediately after the 0.30 upgrade:

All non-deleted contracts will have their expiry extended to at least 90 days after the upgrade date.

Deleted contracts will start to be purged from state; so a `getContractInfo` query that previously\

returned `CONTRACTDELETED` may now report `INVALIDCONTRACTID`.

About 90 days after the 0.30 upgrade, some contracts will begin to expire. The network will try to automatically charge the renewal fee (approximately \$0.026 for 90 days) to the expired contract's auto-renew account. If an auto-renew account has zero balance, the network will then try to charge the contract itself.

A contract unable to pay renewal fees will enter a week-long "grace period" during which it is unusable, unless its expiry is extended via `ContractUpdate` or it receives hbar. After this grace period, the contract will be purged from state.

We strongly encourage all contract authors to set an auto-renew account for their contract. This isolates the contract logic from the existence of rent.

This release also brings two peripheral improvements:

1. It will become possible to schedule a `CryptoApproveAllowance` transaction.
2. Mirror node operators will be able to use the daily `NodeStakeUpdate` export to track the current values of several key staking properties. Please review the linked protobuf comments for more details on these properties.

<figure><figcaption></figcaption></figure>

v0.29

{% hint style="success" %}

MAINNET UPDATE: SEPTEMBER 27, 2022

{% endhint %}

{% hint style="success" %}

TESTNET UPDATE: SEPTEMBER 7, 2022

{% endhint %}

```
{% hint style="success" %}
TESTNET UPDATE: AUGUST 30, 2022
{% endhint %}
```

Contract-managed tokens ▢

In Services 0.29 we have followed the HIP-514 roadmap to give contract authors many new ways to inspect and manage HTS tokens.

The HIP enumerates the ways; examples include a contract that revokes an account's KYC for a token, or deletes a token for which it has admin privileges, or even changes a token's supply key based on the metadata in an NFT!

Note there are four HIP-514 functions that will be part of release 0.30, as follows: `getTokenExpiryInfo(address)`, `updateTokenExpiryInfo(address, Expiry)`, `updateTokenInfo(address, HederaToken)`, `isToken(address token)` and `getTokenType(address token)`.

HIP-435 Record Stream v6 will be enabled on testnet and mainnet in this release.

Deprecations

Please note this important deprecation that will change how clients fetch token associations and balances after the November release in this year. At that time, mirror nodes will become the exclusive source of token association metadata. This is because HIP-367 made token associations unlimited, so in the long run it will not be efficient for consensus nodes to serve this information.

```
<figure><figcaption></figcaption></figure>
```

v0.28

```
{% hint style="success" %}
MAINNET UPDATE: AUGUST 25, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: JULY 29, 2022
{% endhint %}
```

Services 0.28 gives Hedera devs a new dApp building block in HIP-351 (Pseudorandom Numbers). HAPI has a new `UtilService` with a `prng` transaction that generates a record with either a pseudorandom 48-byte seed, or an integer in a requested range.

Smart contracts can also get pseudorandom values by calling a new system contract at address `0x169`, using the interface here as in this example. Applications might include NFT mint contracts, lotteries, and so on.

📖 The HIP-351 text does not yet reflect the name change from `RandomGenerate` to `prng`, or the system contract specification. It does explain in detail how `prng` derives its entropy from the running hash of transaction records generated by the network.

This release also includes some bug fixes and smaller improvements; notably, it:

1. Extends `ContractCallLocal` support to the ERC-20 and ERC-721 functions `allowance`, `getApproved`, and `isApprovedForAll`.
2. Permits staking to contract accounts.

```

```

v0.27

v0.27.7

```
{% hint style="success" %}  
MAINNET UPDATE: AUGUST 9, 2022  
{% endhint %}
```

Any ledger that will grow to billions of entities must have an efficient way to remove expired entities. In the Hedera network, this means keeping a list of NFTs owned by an account, so that when an account expires, we can return its NFTs to their respective treasury accounts.

Under certain conditions in the 0.27.5 release, a bug in the logic maintaining these lists could cause NFT transfers to fail, without refunding fees.

We appreciate the Hedera community working with us on this issue. We invite any users who were affected by this bug to contact support at support@hedera.com.

v0.27.0

```
{% hint style="success" %}  
MAINNET UPDATE: JULY 21, 2022  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: JULY 1, 2022  
{% endhint %}
```

The 0.27 release of Hedera Services initiates the first phase of HIP-406 (Staking). We deeply appreciate the community's feedback on this critical feature!

As wallets and exchanges roll out client support, users will now have the choice to stake their hbar to a node. As nodes accumulate stake, from both individuals and organizations, they will become eligible to pay rewards to their stakers. At this point, once the 0.0.800 account balance has crossed a threshold to be set by the council coin committee, rewards will be permanently activated.

This will set the stage for the second phase of staking, in which a node's contribution to consensus becomes a direct function of its stake, and community nodes with sufficient stake can begin to participate in consensus. Please note the decentralized nature of this process makes it hard to predict exactly when each milestone and phase will be achieved. The immediately visible consequences of the 0.27 release will be simply,

1. The consensus nodes handle CryptoCreate and CryptoUpdate transactions with staking elections---even if not all wallets and exchanges are updated to make these elections just yet.

Observant readers might recall that an earlier alpha release of Services 0.27 also enabled HIP-423 (Long Term Scheduled Transactions). This is a complex feature with some deep implications, and we have decided to defer for one more release before going to production.

v0.26

```
{% hint style="success" %}  
MAINNET UPDATE: JUNE 9, 2022  
{% endhint %}
```

```
{% hint style="success" %}
```

TESTNET UPDATE: MAY 25, 2022
{% endhint %}

In this release, we are excited to deploy support for HIP-410 (Wrapping Ethereum Transaction Bytes in a Hedera Transaction). and HIP-415 (Introduction Of Blocks).

HIP-410 adds a HAPI EthereumTransaction by which an account that was auto-created with an ECDSA(secp256k1) key can submit Ethereum transactions to Hedera by signing with its ECDSA key. (Standard Ethereum restrictions on the sender's nonce apply.) Please see HIP-410 for details, including a summary of some very compelling use cases that the EthereumTransaction enables---for example, "I want to use MetaMask to create a transaction to transfer HBAR to another account".

HIP-415 also anticipates such use cases by standardizing the concept of a Hedera "block"; this is important for a full implementation of the Ethereum JSON-RPC API. The definition is simple: One block is all the transactions in a record stream file. The block hash is the 32-byte prefix of the transaction running hash at the end of the file. And the block number is the index of the record file in the full stream history, where the first file had index 0.

Hedera Services 0.26 implements HIP-376, allowing smart contract developers to use the familiar EIP-20 and EIP-721 "operator approval" with both fungible and non-fungible HTS tokens.

Approved operators can manage an owner's tokens on their behalf; this is necessary for many consignment use cases with third party brokers/wallets/auctioneers.

Any permissions granted in a contract through approve() or setApprovalForAll() have an equivalent HAPI cryptoApproveAllowance or cryptoDeleteAllowance expression---and this expression is externalized as a HAPI TransactionBody in the record stream. That is, the HIP-376 system contracts expose a subset of the native HAPI operations, only within the EVM.

.png>)

v0.25

{% hint style="success" %}
MAINNET UPDATE: MAY 19, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 26, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 21, 202
{% endhint %}

The Hedera Services 0.25 release brings good news for HTS users who manage large numbers of token types, as it delivers HIP-367 (Unlimited Token Associations per Account). In particular, a single account can now serve as treasury for any number of token types. (Please do note the CryptoService HAPI queries still return information for only an account's 1000 most recently associated tokens; mirror nodes remain the best source for full history.)

We are also very excited to announce support for HIP-358 (Allow TokenCreate through Hedera Token Service Precompiled Contract). This HIP supercharges contract integration, making it possible for a smart contract to create a new HTS token---fungible or non-fungible, with or without custom fees. (An interested Solidity developer might consult the examples in this contract.)

In a harbinger of more upcoming HTS precompile support, this release will also enable HIP-336 (Approval and Allowance API for Tokens). Token owners can now approve other accounts to manage their HTS tokens or NFTs, in direct analogy to the `approve()` and `transferFrom()` mechanisms in ERC-20 and ERC-721 style tokens.

Enhancements

HIP-336 implementation #2814
HIP-358 implementation #3015
HIP-367 implementation #2917

Fixes

ERC view functions now usable in `ContractCallLocalQuery` #3061

.png)

v0.24

{% hint style="success" %}
MAINNET UPDATE: APRIL 15, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: APRIL 7, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: MARCH 31, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: MARCH 24, 2022
{% endhint %}

In the 0.24 release of Hedera Services, we are excited to give smart contract developers a new level of interoperability with native Hedera Token Service (HTS) tokens via HIP-218 (Smart Contract interactions with Hedera Token Accounts). The Hedera EVM now exposes every HTS fungible token as an ERC-20 token at the address of the token's 0.0.X entity id; and analogously, every HTS non-fungible token appears as an ERC-721 token. This means a smart contract can look up its balance of a fungible HTS token; or change its behavior based on the owner of a particular HTS NFT. Please see the linked HIP for full details.

This upgrade also creates two new system accounts 0.0.800 and 0.0.801 that will hold reward funds.

One change to the Hedera API (HAPI) is that we now have enough evidence to conclude the experimental `getAccountNftInfos` and `getTokenNftInfos` queries do not have a favorable cost/benefit ratio, and these queries are now permanently disabled.

.jpeg)

v0.23

{% hint style="success" %}
MAINNET UPDATE: MARCH 10, 2022
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE: FEBRUARY 17, 2022
{% endhint %}

Hedera Services 0.23 fleshes out our smart contract service via the implementation of HIP-329 (Support CREATE2 opcode). Smart contract developers are now free to use the CREATE2 EVM opcode. A typical use case is a distributed exchange that wants its pair contracts to have deterministic addresses based on the tokens in the pair.

Please note two issues fixed in this release. First, in release 0.22, the nodes returned the bytes ledgerid stipulated by HIP-33 as a UTF-8 encoding of a hex string. The returned bytes are now the big-endian representation of the ledger's numeric id. Second, prior to this release, the record of a dissociateToken from a deleted token did not list the discarded balance of the dissociated account if the token's treasury was missing. This is now fixed.

.jpeg)

v0.22

```
{% hint style="success" %}
MAINNET UPDATE: FEBRUARY 3, 2022
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE: JANUARY 20, 2022
{% endhint %}
```

The 0.22 release is a paradigm shift for Hedera Services, as we deliver the next major step in our Smart Contracts 2.0 roadmap on the strength of the protean HIP-25, a technical foundation for scaling the world state of our ledger to billions of entities without sacrificing the high TPS enabled by the hashgraph consensus algorithm.

Highlights of this release include:

- Network EVM capacity increased to 15M gas-per-second. (Please see HIP-185 for details.)
- Gas limit per ContractCreate or ContractCall raised to 4M.
- Per-contract storage capacity increased to 10MB.
- Solidity integration with native HTS tokens. (Please see HIP-206 for details.)

We expect more progress in these directions over the coming releases. Do note that the gas usage of the HTS integrations is still evolving; follow this issue to track the finalized gas charges leading up to mainnet release.

There are two other HIP's included in this release not related to the smart contract service. First, HIP-33 enhances queries like CryptoGetInfo with a ledger id that marks which Hedera network answered the query. Second, HIP-31 allows a client to include the expected decimals for a token in a CryptoTransfer. This means a hardware wallet can guarantee its token transactions will have the precision seen by the user in the device display.

While we are gaining momentum in our smart contracts roadmap, we are also deeply committed to improving the developer experience, and welcome issues and ideas in our GitHub repository and Discord!

.jpeg)

v0.21.0

```
{% hint style="success" %}
MAINNET UPDATE: JANUARY 13, 2022
{% endhint %}
```


(1\)%20(1\)%20(1\)%20(1\)%20(1\).jpeg)

v0.19.4

```
{% hint style="success" %}  
MAINNET UPDATE: NOVEMBER 4, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: OCTOBER 28, 2021  
{% endhint %}
```

v0.19.3

```
{% hint style="success" %}  
TESTNET UPDATE: OCTOBER 21, 2021  
{% endhint %}
```

In Hedera Services 0.19, we are thrilled to announce migration of the Hedera smart contract service to the Hyperledger Besu EVM, as laid out in HIP-26. This enables support for the latest v0.8.9 Solidity contracts, and harmonizes our gas schedule with that of the "London" hard fork. The Besu migration also sets the stage for a step change in smart contract performance on Hedera.

Two other HIPs targeting the Hedera Token Service go live in this release. First, the HIP-23 feature set is now enabled, so that any account that has been configured with a non-zero maxAutoAssociations can receive air-drops (i.e., units or NFTs of a token type without explicit association). Second, we have also implemented HIP-24, which provides a safety measure for token types created with a pauseKey. If a TokenPause is submitted with this key's signature, then all operations on the token will be suspended until a subsequent TokenUnpause.

v0.18.1

```
{% hint style="success" %}  
MAINNET UPDATE: OCTOBER 7, 2021  
{% endhint %}
```

In Hedera Services 0.18.1, we have a new scalability profile for NFTs in the Hedera Token Service (HTS). Up to fifty million (50M) NFTs, each with 100 bytes of metadata, may now be minted. Of course our CryptoTransfer and ConsensusSubmitMessage operations are still supported at 10k TPS even with this scale.

In this release, we have also enabled automatic reconnect. This feature comes into play when a network partition causes a node to "fall behind" in the consensus protocol. With reconnect enabled, the node can use a special form of gossip to "catch up" and resume participation in the network with no human intervention. This works even when the node has missed many millions of transactions, and the world state is very different from when it was last active.

We are happy to also announce that accounts can be customized to take advantage of the upcoming HIP-23 (Opt-in Token Associations) feature set. That is, an account owner can now "pre-pay" for token associations via a CryptoCreate or CryptoUpdate transaction, without knowing in advance which specific token types they will use.

Once HIP-23 is fully enabled in release 0.19, when their account receives units or NFT's of a new token type via a CryptoTransfer, the network will automatically create the needed association---no explicit TokenAssociate transaction needed. This supports several interesting use cases; please see the linked HIP-23 for more details.

There are three other points of interest in this release.

First, we have removed the HIP-18 limitations noted in the previous release. The tokenFeeScheduleUpdate transaction has been re-enabled, and multiple royalty fees can now be charged for a non-fungible token type.

Second, the address books in system files 0.0.101 and 0.0.102 will now populate their ServiceEndpoint fields. (However, the deprecated ipAddress, portno, and memo fields will no longer be populated after the next release.)

Third, please note that the TokenService getTokenNftInfos and getAccountNftInfos queries are now deprecated and will be removed in a future release. The best answers to such queries demand historical context that only Mirror Nodes have; so these and related queries will move to mirror REST APIs.

Developers will likely appreciate two other release 0.18.1 items. First, we have migrated to Dagger2 for dependency injection. Second, there is a new getExecutionTime query in the NetworkService that supports granular performance testing in development environments.

.png)

v0.18.0

```
{% hint style="success" %}  
TESTNET UPDATE: SEPTEMBER 23, 2021  
{% endhint %}
```

In Hedera Services 0.18.0, we are happy to announce support for HIP-23 (Opt-in Token Associations). This feature lets an Hedera account owner "pre-pay" for token associations via a CryptoCreate or CryptoUpdate transaction, without knowing in advance which specific token types they will use.

Then, when their account receives units or NFT's of a new token type via a CryptoTransfer, the network automatically creates the needed association---no explicit TokenAssociate transaction needed. This supports several interesting use cases; please see the linked HIP-23 for more details.

There are three other points of interest in this release.

First, we have removed the HIP-18 limitations noted in the previous release. The tokenFeeScheduleUpdate transaction has been re-enabled, and multiple royalty fees can now be charged for a non-fungible token type.

Second, the address books in system files 0.0.101 and 0.0.102 will now populate their ServiceEndpoint fields. (However, the deprecated ipAddress, portno, and memo fields will no longer be populated after the next release.)

Third, please note that the TokenService getTokenNftInfos and getAccountNftInfos queries are now deprecated and will be removed in a future release. The best answers to such queries demand historical context that only Mirror Nodes have; so these and related queries will move to mirror REST APIs.

Developers will likely appreciate two other release 0.18.0 items. First, we have migrated to Dagger2 for dependency injection. Second, there is a new getExecutionTime query in the NetworkService that supports granular performance testing in development environments.

Performance Measurement Results:

v0.17.4

```
{% hint style="success" %}  
MAINNET UPDATE: SEPTEMBER 2, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE: AUGUST 30, 2021  
{% endhint %}
```

In Hedera Services 0.17.2, we are excited to announce support for HIP-17 (Non-fungible Tokens), with a complementary extension to HIP-18 (Custom Hedera Token Service Fees) that lets an NFT creator set a royalty fee to be charged when fungible value is exchanged for one of their creations.

Unique token types and minted NFTs are more natural for many use cases than fungible token types. The Hedera Token Service now supports both natively, so that a single CryptoTransfer can perform atomic swaps with any arbitrary combination of fungible, non-fungible, and h transfers. (Please do note that the "paged" `getAccountNftInfos` and `getTokenNftInfos` queries will remain disabled until release 0.18.0, as several large performance improvements are pending.)

In this release we have made it possible to denominate a fixed fee in the units of the token to which it is attached (assuming the type of this token is `FUNGIBLECOMMON`). Custom fractional fees may now also be set as "net-of-transfer". In this case the recipient(s) in the transfer list receive the stated amounts, and the assessed fee is charged to the sender.

There are a few final points of more specialized interest. First, users of the scheduled transaction facility may now also schedule `TokenBurn` and `TokenMint` transactions. Second, network administrators issuing a `CryptoUpdate` to change the treasury account's key must now sign with the new treasury key. Third, the supported TLS cipher suites have been updated to the following list:

1. `TLSDFHSAWITHAES256GCM SHA384` (TLS v1.2)
2. `TLSSECDHEECDSA WITHAES256GCM SHA384` (TLS v1.2)
3. `TLSAES256GCM SHA384` (TLS v1.3)

⚠ There are two temporary limitations to HIP-18 in this release. First, the `tokenFeeScheduleUpdate` transaction is not currently available. Second, only one royalty fee will be charged for a non-fungible token type. Both limitations will be removed in release 0.18.0.

Performance Measurement Results:

```

```

v0.17.3

```
{% hint style="success" %}  
TESTNET UPDATE: AUGUST 24, 2021  
{% endhint %}
```

Please see 0.17.4 release notes.

v0.17.2

```
{% hint style="success" %}  
TESTNET UPDATE: AUGUST 19, 2021  
{% endhint %}
```

Please see 0.17.4 release notes.

v0.16.1

```
{% hint style="success" %}
```

MAINNET UPDATE COMPLETED: AUGUST 5, 2021
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE COMPLETED: JULY 22, 2021
{% endhint %}

In Hedera Services 0.16.0, we are excited to announce support for HIP-18 (Custom Hedera Token Service Fees).

Hedera tokens can now be created with a schedule of up to 10 custom fees, which are either fixed in units of \hbar or another token; or fractional and computed in the units of the owning token. The ledger automatically charges custom fees to accounts as they send units of a fungible token (or ownership of a NFT, see below) via a CryptoTransfer.

When a custom fee cannot be charged, the CryptoTransfer fails atomically, changing no balances other than for the Hedera network fees.

The five case studies in this document show the basics of how custom fees are charged, and how they appear in records. Note that at most two "levels" of custom HTS fees are allowed, and custom fee-charging cannot require changing more than 20 account balances.

△ There is one variation on custom fees that requires a work-around in this release. Specifically, if a fixed fee should be collected in the units of the "parent" token to whose schedule it belongs, then in Release 0.16.0 this must be accomplished using a FractionalFee as described in this issue. In Release 0.17.0 the more natural FixedFee configuration will be available.

In this release, we have also enabled previewnet support for HIP-17 (Non-fungible Tokens). Unique token types and minted NFTs are more natural for many use cases than fungible token types. The Hedera Token Service will soon support both natively, so that a single CryptoTransfer can perform atomic swaps with any arbitrary combination of fungible, non-fungible, and \hbar transfers.

We are very grateful to the Hedera user community for these interesting and powerful new feature sets.

Performance Measurement Results:

v0.15.1

{% hint style="success" %}
MAINNET UPDATE COMPLETED: JULY 1, 2021
{% endhint %}

{% hint style="success" %}
TESTNET UPDATE COMPLETED: JUNE 17, 2021
{% endhint %}

In Hedera Services 0.15.1, we improved performance and integrated with the latest Platform SDK to enable full support of network reconnect.

These performance improvements let us augment the Hedera world state with records of all transactions handled in the three minutes of consensus time, even when handling 10,000 transactions per second. The HAPI GetAccountRecords query now returns, from state, all such records for which the queried account was the payer account.

We have also finalized the design for the non-fungible token (NFT) support to be added to the Hedera Token Service (HTS) in release 0.16.0. The protobufs for new

HAPI operations are available in the 0.15.0 tag of the hedera-protobufs GitHub repository.\

\

To simplify fee calculations, there is now a maximum entity lifetime of a century for any entity whose lifetime is not \already\ constrained by the maximum auto-renew period. A HAPI transaction that tries to set an expiration further than a century from the current consensus time will resolve to INVALIDEXPIRATIONTIME.

v0.14.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JUNE 3, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: MAY 20, 2021  
{% endhint %}
```

In Hedera Services 0.14.0, we have implemented account auto-renewal according to the specifications of HIP-16. This feature will not be enabled until a later date, after ensuring universal awareness of its impact in the user community.

This release includes notable infrastructure work to enable use of the Platform reconnect feature. Reconnect allows a node that has fallen behind in consensus gossip to catch back up dynamically.

A minor improvement to the Hedera API is that the GetVersionInfo query now includes the optional pre-release version and build metadata fields from the Semantic Versioning spec (if applicable).

To simplify life for system admins who are updating a system account's key, we now waive the signing requirement for the account's new key.

v0.13.2

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: MAY 6, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: APRIL 29, 2021 \[v0.13.2]  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: APRIL 22, 2021 \[v0.13.0]  
{% endhint %}
```

In Hedera Services v0.13.0, we have redesigned schedule transactions. The new design gives collaborating nodes a well-defined workflow if they happen to schedule identical transactions, even if they are using different gRPC client libraries (for example, Go and JavaScript). The new design also reduces the number of signatures required to submit a valid ScheduleSign transaction in many common use cases. Users will be able to schedule CryptoTransfer and ConsensusSubmitMessage transactions in this release. Other transaction types will be introduced in future releases.

```
{% hint style="warning" %}  
Note: The schedule transactions feature will not be enabled in this release;  
it's expected to be enabled on testnet in a subsequent v0.13.2 update on April  
29th. This feature is enabled on previewnet.  
{% endhint %}
```

This release deprecates three fields in the protobuf for system files 0.0.101

and 0.0.102. The three deprecated fields are ipAddress, portno, and memo. When we rely on these fields, we cannot concisely represent node with multiple IP addresses. For example, take mainnet node 0 (account 0.0.3), which as of this writing has proxy IPs 13.82.40.153, 34.239.82.6, and 35.237.200.180. The mainnet 0.0.101 file must include a NodeAddress entry for each proxy, which means duplicating fields like nodeCertHash.

The new protobuf avoid this duplication, letting us represent node 0 in a protobuf equivalent of,

```
{
  "nodeId" : 0,
  "certHash" :
"337390d8fea144afc12e81254a28dac6ea82893836ac072effd85e0a7748580ef28096648c5a7f8
dbb4ce81476815137",
  "nodeAccount" : "0.0.3",
  "serviceEndpoints" : [ {
    "ipAddressV4" : "13.82.40.153",
    "port" : 50211
  }, {
    "ipAddressV4" : "34.239.82.6",
    "port" : 50211
  }, {
    "ipAddressV4" : "35.237.200.180",
    "port" : 50211
  } ]
}
```

However, Services will continue to populate the deprecated fields in duplicate entries for six months, to give all consumers of files 0.0.101 and 0.0.102 time to prepare for exclusive use of the new format. After six months, we will eliminate the duplication and the ipAddress, portno, and memo fields will be left empty. (The fields will never be removed to ensure it remains possible to parse early versions of these system files.)

In a minor point, Services now rejects any protobuf string field whose UTF-8 encoding includes the zero-byte character; that is, Unicode code point 0, NUL. Databases (for example, PostgreSQL) commonly reserve this character as a delimiter in their internal formats, so allowing it to occur in entity fields can make life harder for Mirror Node operators.

To simplify tasks for network admins, we have also streamlined the signing requirements for updates to system accounts, and introduced a Docker-based utility called "yahcli" for admin actions such as updating system files.

v0.12.0

```
{% hint style="success" %}
MAINNET UPDATE COMPLETED: MARCH 12, 2021
{% endhint %}
```

```
{% hint style="success" %}
TESTNET UPDATE COMPLETED: FEBRUARY 26, 2021
{% endhint %}
```

In Hedera Services v0.12.0, we completed the MVP implementation of the Hedera Scheduled Transaction Service (HSTS) as detailed in this design document. This service decouples what should execute on the ledger from when it should execute, giving new flexibility and programmability to users. Note that HSTS operations are enabled on Previewnet, but remain disabled on Testnet and Mainnet at this time.

We have given users of the Hedera Token Service (HTS) more control over the lifecycle of their token associations. In v0.11.0, deleted tokens were immediately dissociated from all accounts. This automatic dissociation no longer occurs. If account X is associated with token Y, then even if token Y is marked for deletion, a `getAccountInfo` query for X will continue to show the association with Y \until\it is explicitly removed via a `tokenDissociateFromAccount` transaction. Note that for convenience, queries that return token balances now also return the decimals value for the relevant token. This allows a user to interpret e.g. `balance=10050` as 100.50 tokens given `decimals=2`.

In a final Hedera API (HAPI) change, we have extended the memo field present on contract and topic entities to the account, file, token, and scheduled transaction entities. (Note this memo is distinct from the short-lived memo that may be given to any `TransactionBody` for inclusion in the `TransactionRecord`.) All of these changes to HAPI are now more easily browsed via GitHub pages here; the new `hashgraph/hedera-protobufs` repository is now the authoritative source of the protobuf files defining HAPI.

Apart from these enhancements to HAPI, the "streams" consumable by mirror node operators now include an alpha version of a protobuf file that contains the same information as the `Balances.csv` files. The type of this file is `AllAccountBalances`.

v0.11.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: FEBRUARY 4, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: JANUARY 26, 2021  
{% endhint %}
```

In Hedera Services v0.11.0, we upgraded the record stream format from v2 to v5 and the event stream format from v3 to v5. These changes are described in detail in the "Record and Event Stream File Formats" article.

We also updated startup code to make the number of system accounts in development and pre-production networks match the number of system accounts on mainnet, creating account numbers 900-1000 on startup if they do not exist.

v0.10.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: JANUARY 7, 2021  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: DECEMBER 17, 2020  
{% endhint %}
```

In Hedera Services v0.10.0, we improved the usability of the Hedera Token Service (HTS) with a new `TotalSupply` field in the receipts of `TokenMint` and `TokenBurn` transactions. Without this field, a client must follow the entire record stream of a token's supply changes to be certain of its supply at the consensus timestamp in the receipt. (Note that HTS operations are now enabled on `Previewnet` and `Testnet`, but remain disabled on `Mainnet` at this time. Please consult the SDK documentation for HTS semantics.)

Also for HTS, we added a property `fees.tokenTransferUsageMultiplier` that scales the resource usage assigned to a `CryptoTransfer` that changes token balances. This scaling factor is expected to be set so that the cost of a `CryptoTransfer` that changes two token balances is roughly 10x the cost of a `CryptoTransfer` that changes only two `hbar` balances.

Apart from HTS, this release drops a restriction on what payer accounts can be used for CryptoUpdate transactions that target system accounts. (That is, accounts with numbers not greater than `hedera.numReservedSystemEntities`.) In earlier versions, only three payers were accepted: The target account itself, the system admin account, or the treasury account. Other payers resulted in a status of `AUTHORIZATIONFAILED`. This entire restriction is removed, with one exception---the treasury must pay for a CryptoUpdate targeting the treasury.

Apart from these functional changes, we fixed an unintentional change in the naming of the crypto balances CSV file, and improved the usefulness of clients under `test-clients/` for testing reconnect scenarios.

v0.9.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: DECEMBER 3, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: NOVEMBER 19, 2020  
{% endhint %}
```

In Hedera Services v0.9.0, we finished the alpha implementation of the Hedera Token Service (HTS). Note that all HTS operations are enabled on Previewnet, but remain disabled on Testnet and Mainnet. Please consult the SDK documentation for HTS semantics.

We made several changes to the HAPI protobuf. First, we removed the deprecated `SignatureList` message type. Second, we added a top-level `signedTransactionBytes` field to the `Transaction` message to ensure deterministic transaction hashes across different client libraries; the top-level `bodyBytes` and `sigMap` fields are now deprecated and the already-deprecated `body` field is removed. Third, we deprecated all fields related to non-payer records, include account send and receive thresholds. This followed from the effective removal of non-payer records in v0.8.1.

For the same reason, the semantics of the `CryptoGetRecords` and `ContractGetRecords` queries have also changed. The only queryable records are now those granted to the effective payer of a transaction that was handled while the network property `ledger.keepRecordsInState=true`. Such records have an expiry of 180 seconds. It is important to note that because a contract account can never be the effective payer for a transaction, any `ContractGetRecords` query will always return an empty record list, and we have deprecated the query.

v0.8.1

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: OCTOBER 22, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: OCTOBER 7, 2020  
{% endhint %}
```

The mainnet release includes the 0.8.0 version updates.

v0.8.0

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: SEPTEMBER 17, 2020  
{% endhint %}
```

In Hedera Services v0.8.0, we made several minor fixes and improvements. This

tag also includes pre-release implementations of several operations for an incipient Hedera Token Service (HTS).

NOTE: HTS operations will remain disabled in non-development environments for some time. These operations are under active development; please consult master for up-to-date semantics.

Enhancements

Deprecated fields related to threshold records in HAPI protobuf #506
Update Receipt proto to pair each Status with NodeID - Receipt is deleted only when the latest (duplicate) transaction expires. getTxRecord API will continue to return ALL records with the transaction ID.
First drafts of tokenCreate, tokenUpdate, tokenDelete, tokenTransfer, tokenFreeze, tokenUnfreeze, tokenGrantKyc, tokenRevokeYc, tokenWipe, and getTokenInfo HAPI operations. #505 and #522

v0.7.0

```
{% hint style="success" %}  
MAINNET UPDATE COMPLETED: SEPTEMBER 8, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPDATE COMPLETED: AUGUST 20, 2020  
{% endhint %}
```

In Hedera Services v0.7.0, we've moved to Swirlds SDK release 0.7.3 which enables zero-stake nodes to be part of a network without affecting consensus. Hedera Services v0.7.0 migrated to new interfaces and methods provided in this version of the Swirlds SDK. HCS topic running hashes are now calculated including the payer account id. The release includes other minor fixes and improvements.

Enhancements

Migrate to Swirlds SDK release 0.7.3 with appropriate settings and logging configurations #347, #427
Update HCS topic running hash to include the payer account id #88
Add zero-stake node functionality #274
Add new stats for the average size of HCS submit message transactions that got handled and for counting the number of platform transactions not created per second #316, #334
Change gRPC CipherSuite to be CNSA compliant #215
Make recordLogPeriod dynamic with a default of 2 seconds #315
Add record with 3-min expiry to effective payer account after handling transaction #348
Enhancements for going open source #378, #379

Documentation changes

Clarify interpretation of response codes UNKNOWN and PLATFORMTRANSACTIONNOTCREATED #314, #394

Bug fixes

Prevent CryptoCreate and CryptoUpdate transactions from giving an account an empty key #58, #60
Fix incorrect submitted smart contract transactions count #371
Validate total ledger balance before starting up Services #258
Add a new rolling file to log all queries with controlled maximum rate #59
Other minor bugs #373

v0.6.0

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: AUGUST 6, 2020  
{% endhint %}
```

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JULY 16, 2020  
{% endhint %}
```

In Hedera Services v0.6.0, we've enhanced the Hedera Consensus Service by supporting HCS Topic Fragmentation. We added, into the `ConsensusSubmitMessageTransactionBody`, an optional field for the current chunk information. For every chunk, the payer account that is part of the `initialTransactionID` must match the Payer Account of this transaction. The entire `initialTransactionID` should match the `transactionID` of the first chunk, but this is not checked or enforced by Hedera except when the chunk number is 1.

Enhancements

- Add support for HCS Topic Fragmentation

Documentation changes

- Protobuf v0.6.0 with HAPI doc update to support HCS Topic Fragmentation

v0.5.8

```
{% hint style="success" %}  
MAINNET UPGRADE COMPLETED: JUNE 18, 2020
```

v0.5.8 includes all of the updates found in v0.5.0
{% endhint %}

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: JUNE 8, 2020  
{% endhint %}
```

Version 0.5.8 includes a patch which addresses the resilience of peer-to-peer networking in the hashgraph consensus platform.

v0.5.0

```
{% hint style="success" %}  
TESTNET UPGRADE COMPLETED: MAY 5, 2020  
{% endhint %}
```

In Hedera Services v0.5.0, we've added TLS for trusted communication with nodes on the Hedera network. For better security, only TLS v1.2 and v1.3 with TLS\ECDHE\ECDSA\WITH\AES\256\GCM\SHA384 and TLS\RSA\WITH\AES\256\GCM\SHA384 cipher suites are allowed.

We've added new metadata in the Hedera NodeAddressBook, accessible in system file 0.0.101. The versions of the node software and gRPC Hedera API (HAPI) are now queryable via `GetVersionInfo` under the new `NetworkService` for node and network-scoped operations.

For Hedera Consensus Service, we've updated the topic running hash calculation to use the SHA-384 hash of the submitted message, rather than the message itself. This reduces the storage requirements needed to validate the hash of a topic. The record of a `ConsensusSubmitMessage` transaction that uses the new hashing scheme will have a new `topicRunningHashVersion` field in its receipt. The value of the field will be 2.

Hedera File Service also has several fixes of note. First, we enabled immutable

files. Second, we relaxed the signing requirements for a FileDelete transaction to match the semantics of a revocation service. Third, we fixed a fee calculation bug that overcharged certain FileUpdate transactions.

For Hedera Smart Contract Service, we've improved visibility into transactions that create child contracts using the new keyword by putting created ids in the record of the transaction; and we now propagate parent contract metadata to created children.

Finally, if you use the throttle properties in system file 0.0.121 to estimate network performance limits, you will also be interested in a new standardized format of those properties. The lists below contain these and other minor updates, bug fixes, and documentation changes.

Enhancements

- Add support for TLS
- Expand address book metadata
- Return all created contract ids
- Propagate creator contract metadata
- Introduce GetVersionInfo query
- Standardize throttle configuration
- Enforce file.encoding=utf-8 on startup
- Make duration properties inclusive for readability

Bug fixes

- Use message SHA-384 hash in running hash
- Enable immutable files
- Relax FileDelete signing requirements
- Fix sbh calculation in FileUpdate
- Return metadata for deleted files
- Enforce receiver signing requirements during contract execution
- Reject invalid CryptoGetInfo
- Reject CryptoCreate with empty key
- Return NOT\SUPPORTED for state proof queries
- Waive fees for 0.0.57 updating 0.0.111
- Waive signing requirements for 0.0.55 updating 0.0.121/0.0.122
- Waive all fees for 0.0.2
- Do not throttle system accounts

Documentation changes

- Replace "claim" with "livehash" as appropriate
- Standardize and clarify HAPI doc

v0.4.1

Software update includes the ability for Hedera to dynamically set throttles on network transaction types.

The following throttles would be updated to: 1000 submit messages per second and 5 topic creates per second.

- Reassigning of new Council Member nodes

v0.4.0

Say hello to the Hedera Consensus Service! This release is the first to include HCS, allowing verifiable timestamping and ordering of application messages.

- Network pricing has been updated to include HCS transactions and queries

Network throttle for HCS set to 1000 tps for submitting messages, and 100 tps for each of the other HCS operations.

- Improved end to end testing.

- General code clean up and refactoring.

- ContractCall - TransactionReceipt response to ContractCall no longer includes

the contractID called

CryptoUpdate - TransactionReceipt response to CryptoUpdate no longer includes the accountID updated

CryptoTransfer - CryptoTransfer transactions resulting in INSUFFICIENT\ACCOUNT\BALANCE error no longer list Transfers in the TransactionRecord transferList that were not applied

Miscellaneous

SDKs

Java SDK has been updated to support the Hedera Consensus Service

JavaScript/Typescript SDK has reached version 1.0.0, supporting all four mainnet services

JavaScript/Typescript SDK supports both running in the browser (with Envoy Proxy) and in Node.

Go SDK now supports all four mainnet services.

Fees

Transfer list within transaction records now shows only a single net amount in or out for each account, reflecting both transfers and any fees paid.

Fixed bug in fee schedule that had resulted in fees for ContractCallLocal, ContractGetBytecode, and getVersion queries being undercharged by \33%

You may get more information regarding transaction record fees here.

SDK Extension Components

The Hedera SDK Extension Components (SXC) is an open sourced set of pre-built components that aim to provide additional functionality over and above HCS to make it easier and quicker to develop applications, particularly if they require secure communications between participants.

Components use the Hedera Java SDK to communicate with the Hedera Consensus Service.

Learn more about Hedera SXC here.

README.md:

description: Join a Hedera Testnet

Testnets

Overview

Hedera test networks provide developers access to a free testing environment for Hedera network services. Testnets simulate the exact development environment as you would expect for mainnet. This includes transaction fees, throttles, available services, etc. To create a Hedera Testnet or Previewnet account, you can visit the Hedera Developer Portal.

Once your application has been built and tested in this test environment, you can expect to migrate your decentralized application (dApp) to mainnet without any changes.

Test

Description
Testnet

Testnet runs the same code as the Hedera Mainnet, designed to provide a pre-production environment for developers about to move to mainnet. You can find compatible SDKs here .
Previewnet

<p>Code that is under development by the Hedera team and likely to be</p>

used in an upcoming release designed to give developers early exposure to features coming down the pipe. Updates to the network are made frequently. There is no guarantee an SDK will readily support the up-and-coming features.

Note: Updates to this network are triggered by a new release and are frequent. These updates will not be reflected on the status page.

Network Service	Availability
Cryptocurrency	Limited
Consensus Service	Limited
File Service	Limited
Smart Contract Service	Limited
Token Service	Limited

Test Network Resets

The mirror node and consensus node test network are scheduled to reset once a quarter. When a testnet reset occurs, all account, token, contract, topic, schedule, and file data are wiped.

Developers will no longer have access to the state data from test network consensus nodes. For example, you will not be able to perform transactions or queries on an account that existed before the reset.

The testnet mirror node will be available for developers to store any data before access is completely removed for two weeks after the date of the reset. You will be able to query old testnet information for the two-week period if it is available.

What you should do:

- Take note of the upcoming reset dates.
- Have the ability to recreate test data for your application to minimize interruptions.
- After the reset, you will need to visit the Hedera Developer Portal to get your new testnet account ID.
- The public and private key pair will remain the same after resets.
- Subscribe to the Hedera status page to receive reset notifications.
- Mirror Node operators can reference the instructions here to set up your mirror node
- GCP GCS and AWS S3 buckets: hedera-testnet-streams-2023-01

If you have any questions or concerns, please connect with us via Discord.

Reset Dates:

2024

February 1, 2024 - Completed
April 25, 2024 - Skipped
July 25, 2024 - Skipped
Oct 31, 2024

2023

January 26, 2023 - Completed
April 27, 2023 - Skipped
July 27, 2023 - Completed
October 26, 2023 - Skipped

Test Network Throttles

{% hint style="warning" %}

Limited Support\

Transactions are currently throttled for testnets. You will receive a BUSY response if the number of transactions submitted to the network exceeds the threshold value.

{% endhint %}

Network Request Type	Throttle (tps)
Cryptocurrency Transactions	<p><code>AccountCreateTransaction</code>: 2 tps</p> <p><code>AccountBalanceQuery</code>: unlimited</p> <p><code>TransferTransaction</code> (inc. tokens): 10,000 tps</p> <p><code>Other</code>: 10,000 tps</p>
Consensus Transactions	<p><code>TopicCreateTransaction</code>: 5 tps</p> <p><code>Other</code>: 10,000 tps</p>
Token Transactions	<p><code>TokenMintTransaction</code>: </p>125 TPS for fungible mint50 TPS for NFT mint<p><code>TokenAssociateTransaction</code>: 100 tps
<code>TransferTransaction</code> (inc. tokens): 10,000 tps
<code>Other</code>: 3,000 tps</p></td></tr><tr><td>Schedule Transactions</td><td><code>ScheduleSignTransaction</code>: 100 tps
<code>ScheduleCreateTransaction</code>: 100 tps</td></tr><tr><td>File Transactions</td><td>10 tps</td></tr><tr><td>Smart Contract Transactions</td><td><code>ContractExecuteTransaction</code>: 350 tps
<code>ContractCreateTransaction</code>: 350 tps</td></tr><tr><td>Queries</td><td><code>ContractGetInfo</code>: 700 tps
<code>ContractGetBytecode</code>: 700 tps
<code>ContractCallLocal</code>: 700 tps
<code>FileGetInfo</code>: 700 tps
<code>FileGetContents</code>: 700 tps
<code>Other</code>: 10,000 tps</td></tr><tr><td>Receipts</td><td>unlimited (no throttle)</td></tr></tbody></table></p>

testnet-access.md:

Testnet Accounts

You will need a Hedera Testnet or Previewnet account to interact with and pay for any network services (cryptocurrency, consensus, tokens, files, and smart contracts). Your Hedera Testnet account holds a balance of HBAR for transfers to other accounts or payments for network services.

Step 1: Create Hedera Portal Profile

To create your Hedera Portal profile, register here and complete your profile. Once you've completed setting up your profile, select the test network (Testnet or Previewnet) from the network drop-down menu and create an account. After account creation, your portal account will automatically receive 1000 HBAR.

You can easily copy your accountId, public key, and private key information to your clipboard to use when configuring your SDK environment for testnet.

{% hint style="info" %}

Note: When previewnet or testnet is reset, new account IDs will be generated. The public and private key pair remain consistent during previewnet and testnet resets. If you receive an invalid account ID response from the network it is likely you need to update your previewnet or testnet account ID. Create an Personal Access Token/API key to streamline the process of account recreation and management when there is a network reset.

{% endhint %}

You're now ready to build your application on testnet!

testnet-nodes.md:

Testnet Consensus Nodes

Testnet nodes belong to the test network and run the same code, or an updated version, as the Hedera Mainnet nodes. Please visit the Hedera status page for the latest versions supported on each network.

Node ID	Node Account ID	Node IP Address	Port
0	0.0.3	0.testnet.hedera.com	34.94.106.61 50.18.132.211
1	0.0.4	1.testnet.hedera.com	35.237.119.55 3.212.6.13
2	0.0.5	2.testnet.hedera.com	35.245.27.193 52.20.18.86
3	0.0.6	3.testnet.hedera.com	34.83.112.116 54.70.192.33
4	0.0.7	4.testnet.hedera.com	34.94.160.4 54.176.199.109
5	0.0.8	5.testnet.hedera.com	34.106.102.218 35.155.49.147
6	0.0.9	6.testnet.hedera.com	34.133.197.230 52.14.252.207

You will need to create your Hedera portal profile to receive a testnet account ID.

```
{% content-ref url="testnet-access.md" %}  
testnet-access.md  
{% endcontent-ref %}
```

Testnet Node Public Keys

Below, you will find the testnet node public keys. The public keys stored in the address book file are hex-encoded (x509). The testnet address book file ID is 0.0.102.

Node Account ID	Public Key
0.0.3	308201a2300d06092a864886f70d01010105000382018f003082018a02820181009f1f8a121c2fd6c76fd508d3e429f0c64bcb44c82a70573552aadcad071569e721958f5a5d09f9587ffa9cfbe5341a2f0114acae346ef3c90213d3436ebb27f4350c990c5c8c3f8e1e36707bc08d42560823e3f24e09a03ad0955a5098019629dd04b27b251dce055f3ddcb0a41d66f0941b0b87cdfc3498d46038ab5df06f62a5ade08598573a88c8f5860dc1492a6e186485a9b13250e6d17b80cd39c5c819109e73ca732db23ef8baa776ec85ce0091becb2edefbaa5ed3e5dbfbd1f885a4fa881af3f144a8a565853533d89393592086b2d1d362e45bfe1fb45683aba6c640979ad6b46877184726c6ebd58b2eae85c7cfe3f3babef5f6cced850034b3847206c2d678c361876026b8d351e002af5e0ffe6f5b1f295fdc2f469caa2d2381ea0b48ca987cc2c8e635e8b19ce5e172a93761a8d490a9a4518d7255880a14d77b7ba774892b92a40bb81362e34fc6d5178d9b30112934205cb77fb9a282427394564a8554ea47286a47f86239e75c94789ce98c99844782462944f613167d7b50203010001
0.0.4	308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c557af579fa83501be899b28907765bdfdc52ab432b0195a1f1ecd86fc00ab6c5509b0fdd97edd3cb5cea56a295f312abb550831dbf963f450118b4fcc6e22cf4676200ce9cc8edfbbf558dc69f024264ad7d3dab23bed2133c274e6934489155db1087f90370905c64185a6211dc742fb9a6909d82186947b277463dfb3ff0acd47eff12ead1f6972ef2c1203793c45e77575be4fa110c7e40fa8db9c6187d113f4704014179071abf59be7d2b0de82de4215dc25506b1c9c26e4917401c997506e377e6bf03b688727e7940fad69c5e0da3cd5cbd2be777350aea2d0d47e97a448c84be6ce134d64bee0985c29162f4c1e567cca93d06a3c1be8abce35b557fb77f4fe671a66dec790756d0e8818165f2bacaa891aae7ac7437f

c7175b6eb6deb7472378751bb6bf9b0e1483f9668e9fdbd5604c39b14d9e2bedeec846a980d704d171e7ba4b7fcd1a30d945ca12f47a325d9398aa18f97066054d4d15fc8994e2debe73e9271d548683f61ea44fb25071e3518a78ed3eb37e71a0691f2670203010001</td></tr><tr><td>0.0.5</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a02820181009ba457b73305f04a91cc46b1b965c4e841751abc8b1415a0badfd1f32c2482386a22725eb7ec74dea21e50617d648ea5ac393741ab01b8efb321239b8d4fdb1dfbeb9e3f39aa46580dd045d18ca44d002c37ddb527cce4ddc32bfc73419671f4ca4464a3f2a84fc85c71acf0e5a89626df69a81474ed16529f801a8afa97e435c4e04a964a357527288843e58f0a05cf5153ee4507b2c68b3d7fb54ae6a95a959c87a12f630e95c7b1b3c3695e858662417926d76c16983faf61225038745907e9cf13d67c2acd503ca451c85933ac4118acc279801cb968349903145ced27629dd08916317093587a77c2205cfa52543b53c3b6ea15b84e3d2c30c1ed752a4633c36b25b9893ea02ad562eb9b7868b3b4f47f4a25e356064962ac7b25e582944f00d30798a262f9214d8c5e74d0a8376cc2d6ba64e18f5e4a40afac625062d2ca23cd2800708321d3834314f0e5844859232673a32e70ae0d711e310581bcd14e87134694c6e0930f46b37b96d49a64573947331e7e507d9e56de5e6146f2f0203010001</td></tr><tr><td>0.0.6</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c42ccac5fbc691fbbbebda87ffd1e75bdcd8922494cf44fdbccce49788521c378bf77db0934ec0d2183d7c51db66f864c11ab7de1ac3c4cfdc1f093a2d6f37e2b34cbe4c8131f9683ad42878c83d3554c645aa167bcfb064a83dc45c5b1158499f9d92587fff7abcd5f221cd8150548413000fa6e5659089b1dfd65766ea78eaeafca6b45455fd8ab5984dbe35e5795d2c635ea7974d43e8eae4febffe492e707b48b1b0fc6481ae9e09d39133009b7d26402e6e52e5e91b2b380d88f0be7fb4b303e70219785057aa94ce924c4926e916569286e86b3ba651ca2a0a63df4f6907fefe3483d93b4ce1d4d03c7142111375b2c2c51d4eb839e37af530b2cbd6f50d4cb36e27937170d9cddac0ace2cc24b804b0a27351cf830b76525e26dfb9dbf49a056624a76862494e7263d0d70cebae952943e55842f5cad13fcf60a2e6dcf7a1d533f3a5bb54ec21918c76e525ba29146675831e17e36c61fe85498828d09b762015412b2e527849baec1cfff77de4c294c550811e598ff24da15a34569dd0203010001</td></tr><tr><td>0.0.7</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100902f0490a9b7f5d2cd1c0d96c6a6990f573b5f0eb5bdbba39661ef023092419344669969a68a4c7071d329990fb1792e9001cb5598ea71c2d6676824320ee4cabf1dd357ae7f2adbedc1b1b0a9d95623779b4c4c7b47c4787a16ee7188c7217177624a9264ab39c41f7ff0b45a89bda40c4ad07c4d596d5f09d7056bcb5a35f44f95a59c266e09892dcbe46ad51f2d2b3e991a8f6658e1f2cb94c773eb44c44e892d1e55c1076f1608319ee657e40f192967543ab42ab222386d17586e253748dab025e50b50ae6050720e239d64ee6fb4507c0614dd4be7afdb1330890ffa3a6e176527c3116af129a9ac5e336d9f601e7127a6d7d820ad2f902dac9b248668a1bab08d10342ea69a7097132ff7120cc64fcde7840c656ba1732ba95e9c36751175e4ec3d84a7e0d28842b41bbbbd6f28e46c3a6633e1827965c55820d50dae2b0465cc0d42e195b9d1532e6225eb998d6a49079a8a1cd4d0175de3c87f97614847b3cbb17aa34be820b7b3ad98ac3faef993a6778974782c0c4ae3fabbbcc430203010001</td></tr><tr><td>0.0.8</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810091d7dff7f784efbfe5890450c5bc9e3534bffadad93fb7afb15bc7bcf67d3d3b413bd99940dd82564ada04ab2e4edf0a1c0b8fb7e1a8092e9138e960be2cc68b5b97f57d281c5872e97a479fc848363160e3863b57b33e4869b185ace5e36bd43ae5fa678c9eb66f1f4014786826b2f8fa7e0060f4405c0a8f9da7205ff4683a243fa0f315f1afbb4a4d140d02234e4473fb92fcb38f3eb28c60cf7cbfb64e069c18086e4dd61938920ae0fd7c193e6e104e65b817ed9398e232237fdf08322c9cec09d4099272a7c015d22b4dcc969f6ea1f518902105df60092b55a41b4f32b957b57d84e5b223905e8698951733ea9f2e2461ec0d6522ee816d5850facfeb412cff9b99943a87dc0d046447ce93b97e16d73b96b4263962f81fcf9458e57577c780a6f1615aa7a12326738e269bb731f89e891622e577ea54420bf0ca46be6fc4f71cf2681ac0252aa885e13be672cd284590427dcd137cf311625e8bee3b08fdcaaf465b387ce7cb33816f2c14a6b99ac7d734318cfc59b7ed939bafef8790203010001</td></tr><tr><td>0.0.9</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c6e18c8fbf4cd4eb104542cb20aaaa252d95f052f1086d581c44ad737bf6676c0c3f789af5265b8afb79b50912da84e0afcf7547cb1fff08d0527017eb6dc5cdf83b51969d44336a6387cd70b94bf4c9baf2029840e5f4f863d7081f0fa81e0863adedb8b89a5dac2bb552d6e7b9fba222ac28c57075538fc957992942d341fa2876e6b507e9ce7ed572e8cfda5defa364fdf8d8e23829a4ccbb478f11eee3b32ab85e072951c5d9420115fba327073494f43b5f6bebf84152e356e7b16ba764b7a3b52cb2734640163be1465e6d1fa4c6e6f66684a635c9a556aa7100dbe645df8f4c423ae45a08cb35b4bc187886e2299b5c0210a5fba3b9449f483ef94ed922e1e98c113be166b89c73582243135d442306abe5a71b77018ff335d6dd79542697b168238b96727fd1339b5f82a3b6a597d976037ae2506456c8b34e9fbf3bc32410441c4bfc8eba58597254efebfaa78809a5c8854729a5ba78ece19fc8407dd8894a6bc7844037d878cace6c152c2e89e8a64b068a6c237e09993be806890203010001</td></tr></tbody></table>

Preview Testnet Nodes

Preview testnet nodes belong to the preview test network. The preview testnet nodes run code that is currently under development as an early preview for users. This network is unstable and is not recommended as a suitable pre-production environment.

Node ID	Node Account ID	Node IP Address	Port
0	0.0.3	0.previewnet.hedera.com	35.231.208.148
1	0.0.4	1.previewnet.hedera.com	3.211.248.172
2	0.0.5	2.previewnet.hedera.com	3.133.213.146
3	0.0.6	3.previewnet.hedera.com	35.225.201.195
4	0.0.7	4.previewnet.hedera.com	52.15.105.130
5	0.0.8	5.previewnet.hedera.com	54.241.38.1
6	0.0.9	6.previewnet.hedera.com	35.235.65.51
7	0.0.10	7.previewnet.hedera.com	35.83.89.171
8	0.0.11	8.previewnet.hedera.com	34.106.247.65
9	0.0.12	9.previewnet.hedera.com	35.234.125.23

You will need to create and complete your Hedera portal profile to receive a previewnet account ID.

```
{% content-ref url="testnet-access.md" %}  
testnet-access.md  
{% endcontent-ref %}
```


Previewnet Node Public Keys

Below, you will find the previewnet node public keys. The public keys stored in the address book file are hex encoded keys (x509). The previewnet address book file ID is 0.0.102.

Node Account ID	Public Key
0.0.3	308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c557af579fa83501be899b28907765bfdfcd52ab432b0195a1f1ecd86fc00ab6c5509b0fdd97edd3cb5cea56a295f312abb550831dbf963f450118b4fcc6e22cf4676200ce9cc8edfbbf558dc69f024264ad7d3dab23bed2133c274e6934489155db1087f90370905c64185a6211dc742fb9a6909d82186947b277463dfb3ff0acd47eff12ead1f6972ef2c1203793c45e77575be4fa110c7e40fa8db9c6187d113f4704014179071abf59be7d2b0de82de4215dc25506b1c9c26e4917401c997506e377e6bf03b688727e7940fad69c5e0da3cd5cbd2be777350aea2d0d47e97a448c84be6ce134d64bee0985c29162f4c1e567cca93d06a3c1be8abce35b557fb77f4fe671a66dec790756d0e8818165f2bacaa891aae7ac7437fc7175b6eb6deb7472378751bb6bf9b0e1483f9668e9fdbd5604c39b14d9e2bedeec846a980d704d171e7ba4b7fcd1a30d945ca12f47a325d9398aa18f97066054d4d15fc8994e2debe73e9271d548683f61ea44fb25071e3518a78ed3eb37e71a0691f2670203010001
0.0.4	308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c557af579fa83501be899b28907765bfdfcd52ab432b0195a1f1ecd86fc00ab6c5509b0fdd97edd3cb5cea56a295f312abb550831dbf963f450118b4fcc6e22cf4676200ce9cc8edfbbf558dc69f024264ad7d3dab23bed2133c274e6934489155db1087f90370905c64185a6211dc742fb9a6909d82186947b277463dfb3ff0acd47eff12ead1f6972ef2c1203793c45e77575be4fa110c7e40fa8db9c6187d113f4704014179071abf59be7d2b0de82de4215dc25506b1c9c26e4917401c997506e377e6bf03b688727e7940fad69c5e0da3cd5cbd2be777350aea2d0d47e97a448c84be6ce134d64bee0985c29162f4c1e567cca93d06a3c1be8abce35b557fb77f4fe671a66dec790756d0e8818165f2bacaa891aae7ac7437fc7175b6eb6deb7472378751bb6bf9b0e1483f9668e9fdbd5604c39b14d9e2bedeec846a980d704d171e7ba4b7fcd1a30d945ca12f47a325d9398aa18f97066054d4d15fc8994e2debe73e9271d548683f61ea44fb25071e3518a78ed3eb37e71a0691f2670203010001
0.0.5	308201a2300d06092a864886f70d01010105000382018f003082018a02820181009ba457b

73305f04a91cc46b1b965c4e841751abc8b1415a0badfd1f32c2482386a22725eb7ec74dea21e506
17d648ea5ac393741ab01b8efb321239b8d4fdb1dfbe9e3f39aa46580dd045d18ca44d002c37ddb
527cce4ddc32bfc73419671f4ca4464a3f2a84fc85c71acf0e5a89626df69a81474ed16529f801a8
afa97e435c4e04a964a357527288843e58f0a05cf5153ee4507b2c68b3d7fb54ae6a95a959c87a12
f630e95c7b1b3c3695e858662417926d76c16983faf61225038745907e9cf13d67c2acd503ca451c
85933ac4118acc279801cb968349903145ced27629dd08916317093587a77c2205cfa52543b53c3b
6ea15b84e3d2c30c1ed752a4633c36b25b9893ea02ad562eb9b7868b3b4f47f4a25e356064962ac7
b25e582944f00d30798a262f9214d8c5e74d0a8376cc2d6ba64e18f5e4a40afac625062d2ca23cd2
800708321d3834314f0e5844859232673a32e70ae0d711e310581bcd14e87134694c6e0930f46b3
7b96d49a64573947331e7e507d9e56de5e6146f2f0203010001</td></tr><tr><td>0.0.6</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c42ccac
5fbc691fbbabda87ffd1e75bdcd8922494cf44fdbccee49788521c378bf77db0934ec0d2183d7c51
db66f864c11ab7de1ac3c4cfdc1f093a2d6f37e2b34cbe4c8131f9683ad42878c83d3554c645aa16
7bcfb064a83dc45c5b1158499f9d92587fff7abcd5f221cd8150548413000fa6e5659089b1dfd657
66ea78eaeafca6b45455fd8ab5984dbe35e5795d2c635ea7974d43e8eae4febffe492e707b48b1b0
fc6481ae9e09d39133009b7d26402e6e52e5e91b2b380d88f0be7fb4b303e70219785057aa94ce92
4c4926e916569286e86b3ba651ca2a0a63df4f6907fefe3483d93b4ce1d4d03c7142111375b2c2c5
1d4eb839e37af530b2cbd6f50d4cb36e27937170d9cddac0ace2cc24b804b0a27351cf830b76525e
26dfb9dbf49a056624a76862494e7263d0d70cebae952943e55842f5cad13fcf60a2e6dcf7a1d533
f3a5bb54ec21918c76e525ba29146675831e17e36c61fe85498828d09b762015412b2e527849baec
1cfff77de4c294c550811e598ff24da15a34569dd0203010001</td></tr><tr><td>0.0.7</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100902f049
0a9b7f5d2cd1c0d96c6a6990f573b5f0eb5bdbba39661ef023092419344669969a68a4c7071d3299
90fb1792e9001cb5598ea71c2d6676824320ee4cabf1dd357ae7f2adbedc1b1b0a9d95623779b4c4
c7b47c4787a16ee7188c7217177624a9264ab39c41f7ff0b45a89bda40c4ad07c4d596d5f09d7056
bcb5a35f44f95a59c266e09892dcbe46ad51f2d2b3e991a8f6658e1f2cb94c773eb44c44e892d1e5
5c1076f1608319ee657e40f192967543ab42ab222386d17586e253748dabd025e50b50ae6050720e
239d64ee6fb4507c0614dd4be7afdb1330890ff3a6e176527c3116af129a9ac5e336d9f601e7127a
6d7d820ad2f902dac9b248668a1bab08d10342ea69a7097132ff7120cc64fcde7840c656ba1732ba
95e9c36751175e4ec3d84a7e0d28842b41bbbd6f28e46c3a6633e1827965c55820d50dae2b0465c
c0d42e195b9d1532e6225eb998d6a49079a8a1cd4d0175de3c87f97614847b3cbb17aa34be820b7b
3ad98ac3faef993a6778974782c0c4ae3fabbcc430203010001</td></tr><tr><td>0.0.8</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a028201810091d7dff
f78f4efbe5890450c5bc9e3534bffadad93fb7afb15bc7bcf67d3d3b413bd99940dd82564ada04ab
2e4edf0a1c0b8fb7e1a8092e9138e960be2cc68b5b97f57d281c5872e97a479fc848363160e3863b
57b33e4869b185ace5e36bd43ae5fa678c9eb66f1f4014786826b2f8fa7e0060f4405c0a8f9da720
5ff4683a243fa0f315f1afbb4a4d140d02234e4473fb92fcb38f3eb28c60cf7cbfb64e069c18086e
4dd61938920ae0fd7c193e6e104e65b817ed9398e232237dfdf08322c9cec09d4099272a7c015d22b
4dcc969f6ea1f518902105df60092b55a1b4f32b957b57fd84e5b223905e8698951733ea9f2e2461
ec0d6522ee816d5850facfeb412cff9b99943a87dc0d046447ce93b97e16d73b96b4263962f81fcf
9458e57577c780a6f1615aa7a12326738e269bb731f89e891622e577ea54420bf0ca46be6fc4f71c
f2681ac0252aa885e13be672cd284590427dcd137cf311625e8bee3b08fdcaaf465b387ce7cb3381
6f2c14a6b99ac7d734318cfc59b7ed939bafef8790203010001</td></tr><tr><td>0.0.9</td><td>308201a2300d06092a864886f70d01010105000382018f003082018a0282018100c6e18c8
fbf4cd4eb104542cb20aaaa252d95f052f1086d581c44ad737bf6676c0c3f789af5265b8afb79b50
912da84e0afc7547cb1fff08d0527017eb6dc5cdf83b51969d44336a6387cd70b94bf4c9baf2029
840e5f4f863d7081f0fa81e0863adedb8b89a5dac2bb552d6e7b9fba222ac28c57075538fc957992
942d341fa2876e6b507e9ce7ed572e8cfda5defa364fdf8d8e23829a4ccbb478f11eee3b32ab85e0
72951c5d9420115fba327073494f43b5f6bebf84152e356e7b16ba764b7a3b52cb2734640163be14
65e6d1fa4c6e6f66684a635c9a556aa7100dbe645df8f4c423ae45a08cb35b4bc187886e2299b5c0
210a5fba3b9449f483ef94ed922e1e98c113be166b89c73582243135d442306abe5a71b77018ff33
5d6dd79542697b168238b96727fd1339b5f82a3b6a597d976037ae2506456c8b34e9fbf3bc324104
41c4bfc8eba58597254efebfaa78809a5c8854729a5ba78ece19fc8407dd8894a6bc7844037d878c
ace6c152c2e89e8a64b068a6c237e09993be806890203010001</td></tr></tbody></table>

Test Network Mirror Nodes

For test network mirror node information, please visit the Hedera Mirror Node section below  .

```
{% content-ref url="../../../core-concepts/mirror-nodes/hedera-mirror-node.md" %}
hedera-mirror-node.md
{% endcontent-ref %}
```

pyth-network-oracle.md:

Pyth Oracles

What is Pyth?

The Pyth Network is a first-party financial oracle network designed to provide low-latency real-world data to multiple blockchains securely and transparently. Pyth currently supports 400+ real-time price feeds across crypto, equities, ETFs, FX pairs, and commodities and has facilitated more than \$100B in total trading volume across over 50 blockchain ecosystems.

How to integrate with the Pyth Network on Hedera?

The Pyth Price Feeds uses a "pull" price update model, where users are responsible for posting price updates on-chain when needed. In the pull model, developers should integrate Pyth into both their on-chain and off-chain code:

1. On-chain programs should read prices from the Pyth program deployed on the same chain
2. Off-chain frontends and jobs should include Pyth price updates alongside (or within) their application-specific transactions.

Please note that it is possible to replicate the legacy Oracle design with the Pyth scheduler (previously known as "price pusher"). It is an off-chain application that regularly pulls price updates on to a blockchain you can find [here](#).

Demonstration: How to integrate with the Pyth Network on Hedera?

This demo is a simple example of using Pyth prices in Hedera and is based on the tutorial [here](#). Follow the instructions in the tutorial to build, deploy, and use this demo.

This demo is similar to the contract written in the tutorial. The only difference is that it uses a different math to calculate the price of 1\$ in HBAR because, In the Hedera EVM layer, the native token has only 8 decimal places, while Ethereum has 18 decimal places. This means that the smallest unit of HBAR (1 wei in Hedera EVM) is 0.00000001 HBAR, while the smallest unit of ETH is 0.000000000000000001 ETH. You can see the change in the code in the `MyFirstPythContract.sol`.

```
{% embed url="https://youtu.be/2ry0sTvnGo" %}
```

For more details, please visit this [GitHub repository](#).

If you have any questions, please refer to the [Pyth Network documentation](#) or join [Discord](#).

Contributors: @KemarTiti

README.md:

Open Source Solutions

supra-oracles.md:

Supra Oracles

Overview

Supra is a novel, high-throughput Oracle & IntraLayer offering a vertically integrated toolkit of cross-chain solutions. These solutions include data oracles, asset bridges, and automation networks that aim to interlink all public and private blockchains. Supra provides decentralized Oracle price feeds to deliver real-world data to web3 ecosystems through various on-chain and off-chain use cases. Oracles ensure that the data from the real world is accurate, which is crucial for decentralized applications (dApps) that rely on real-world, real-time data. This is important for dApps that need real-time information, such as the prices of cryptocurrencies and various other assets. The Supra x Hedera integration aims to bring speed, security, and accuracy to real-time data feeds, enhancing the functionality and reliability of dApps on the Hedera network.

Developer Considerations

Contract calls, such as ethcall and ethestimateGas, go to the mirror node, which limits those to a small data payload size. Our engineering team has successfully upgraded the API call data payload capacity to 24 KB. This enhancement is designed to fetch a single price pair from the data feed efficiently, ensuring a more streamlined retrieval process.

{% hint style="info" %}

Please Note: While this update offers improved performance to pull single price pairs, attempting to pull more than one price pair at a time may surpass the new 24 KB data payload limit. Should this limit be exceeded, the API will return an error message. We recommend structuring your API calls accordingly to avoid any potential disruptions.

{% endhint %}

☞	https://supra.com/docs/readme/	Official Documentation
🖼️	supra-oracles7134.jpg	
📄	https://supra.com/docs/readme/	
📊	https://supra.com/data/catalog/details?instrumentName=hbarusdt&#x26;providerName=supra	HBAR Price Feed
🖼️	hedera-logo-black (1).png	
📄	https://supra.com/data/catalog/details?instrumentName=hbarusdt&#x26;providerName=supra	

dao-proposals.md:

DAO Proposals

To create a proposal on chain, you must have sufficient voting power. This information can be found on the DAO's Overview tab from the HashioDAO dashboard.

{% hint style="info" %}

Note: There are no snapshots, like Ethereum-based DAOs, instead, your tokens are locked up for the duration of the vote. Your voting power is equal to the amount of tokens you lock up.
{% endhint %}

Click on Manage and lock up tokens to increase your voting powers.

<figure><figcaption></figcaption></figure>

Create a Proposal

From the HashioDAO dashboard, click on the DAO you created in the previous step to manage the DAO and create new proposals. Click New Proposal and let's review the different types of proposals and examples. You can create a proposal for various activities like a text, token transfer, token associate, and upgrade DAO proposal.

<figure><figcaption></figcaption></figure>

<details>

<summary>Text Proposal Example</summary>

This Text proposal transaction is proposing to create a newsletter to go out to members of the DAO on a weekly basis.

1. Click on New Proposal
2. Fill out the title, description, link to discussion, and optionally a custom markdown description
3. Click Submit and approve the create proposal transaction in your wallet

</details>

<details>

<summary>Token Transfer Proposal Example</summary>

This Token Transfer proposal is proposing to transfer the specified token to an account.

1. Click on New Proposal
2. Fill out the title, description, link to discussion, recipient account ID, and select the token to transfer and the amount.
3. Click Submit and approve the create proposal transaction in your wallet

</details>

<details>

<summary>Token Associate Proposal Example</summary>

This Token Associate proposal transaction is proposing the HashioDAO token to be associated with the smart contract. Once the proposal is created, members of the DAO can vote on it to execute or reject the proposal.

1. Click on New Proposal

2. Fill out the title, description, link to discussion, and asset/token ID you want to associate
3. Click Submit and approve the create proposal transaction in your wallet

</details>

Vote on Proposals

To vote on a proposal, first ensure that your wallet is connected and have voting powers. Navigate to the Proposals tab on the DAO's dashboard and choose one of the active proposals to vote on.

<figure><figcaption></figcaption></figure>

Cast your vote and approve the transaction in your wallet to confirm your vote.

<figure><figcaption></figcaption></figure>

Execute Proposal

After the voting period ends, click on a proposal that passed under the Proposals tab and click Execute.

<figure><figcaption></figcaption></figure>

Conclusion and Additional Resources

Congrats on successfully creating, voting, and executing your first DAO proposal!

- [HashioDAO Repository](#)
- [HashScan Network Explorer](#)
- [HashioDAO Web Application](#)

governance-token-dao.md:

Governance Token DAO

Token DAO Creation

After creating and funding a Hedera account, navigate to the HashioDAO web application dashboard and connect your wallet. Once your wallet is connected, click Create a new DAO and this will guide you through a series of setup steps where you will define the details of your DAO including name and governance type.

<figure><figcaption></figcaption></figure>

Read and accept the disclaimer and click Next.

<figure><figcaption></figcaption></figure>

Define the details of your DAO by filling out each field. These details are important because it creates your DAO's public profile and how the community will be identified.

<figure><figcaption></figcaption></figure>

Governance Models

Choose the governance model that works best with your organizations's goals. There are three models to choose from: governance token, multisig, and NFT.

<figure><figcaption></figcaption></figure>

The createDAO() function is called on the FTDaoFactory factory contract when you create a token DAO. This initiates a new DAO treasury smart contract based on the parameters you set for you token. These parameters will be set in the next step.

Define Governance Token and Voting Details

Define the DAO's governance token and voting details. You have the option to create a new governance token or use an existing one. For the purposes of this tutorial, we will create a new one. Define the details for your new governance token:

<figure><figcaption></figcaption></figure>

Once you click Create Token and approve the transaction in your wallet a Governor smart contract is deployed for managing proposals, voting, and executing decisions. This contract streamlines proposal handling and ensures a fair voting process based on community consensus. This transaction will also generate a token ID for your governance token.

<figure><figcaption></figcaption></figure>

Set the voting quorum percentage, voting duration, and delay period for votes. To avoid delays, starting with a low quorum percentage is recommended. This can be adjusted over time as the community's needs change.

{% hint style="info" %}

DAOs set their quorum percentage depending on their goals and needs. If the quorum is set too high, decisions are only made with a large number of participation from the members. If the quorum is set low, this allows more flexibility in decision making and the chances of proposals being approved would be higher than with a higher quorum.

{% endhint %}

<figure><figcaption></figcaption></figure>

{% hint style="warning" %}

Be cautious when depositing governance tokens in to the treasury. Since the treasury does not have voting rights, depositing a large portion of your governance tokens may result in being locked out of creating and voting on proposals and accessing treasury funds. Always check that these allocations don't limit your ability to reach a voting quorum.

{% endhint %}

Review and Submit to Create New DAO

Review the DAO details, governance type, and submit the transaction. Approve the DAO creation transaction in your wallet to proceed.

<figure><figcaption></figcaption></figure>

Your new DAO should show up on the HashioDAO dashboard once the transaction is approved and confirmed. If the DAO was successfully creation, a "Created new public Governance Token DAO \\\\" message will pop-up on the top-right corner of your browser (see below for example).

<figure><figcaption></figcaption></figure>

Lastly, send in some HBAR to the treasury to finish setting up the DAO by clicking on the Deposit button under the Assets tab. Approve the transaction in your wallet.

{% hint style="info" %}

When a DAO is created, its financial resources are managed through the DAO treasury smart contract. This contract facilitates the distribution of funds by allowing transactions to be initiated by the admin. In this structure, the role of the admin is fulfilled by the voting (Governor) contract.

{% endhint %}

<figure><figcaption></figcaption></figure>

Conclusion and Additional Resources

Congrats on successfully creating your first DAO using the governance token model! In this guide, you defined and customized the details of the new DAO, governance token, and voting processes. Happy DAO creating!

[➞ HashioDAO Repository](#)

[➞ HashScan Network Explorer](#)

[➞ HashioDAO Web Application](#)

[➞ What is a Governance Token?](#)

local-environment-setup.md:

Local Environment Setup

This tutorial will help get your local environment setup and show you how to secure your setup for the local wallet pairings. You will also configure

environment variables to use the Pinata IPFS API. By the end of this guide, you'll have your local environment set up and configured to run the HashioDAO application locally.

Prerequisites

- The Vercel CLI installed.
- A Pinata account created.
- Git command line and TypeScript >= 4.7 installed.

Step 1: Project Installation

Open a new terminal and navigate to your preferred directory where you want your project to live. Clone the repo and install dependencies using these commands:

```
bash
git clone https://github.com/hashgraph/hedera-accelerator-defi-dex-ui.git
yarn install
```

These commands clone the project repository onto your local machine and install all the necessary dependencies using the yarn package manager.

Step 2: Local Environment Setup

Setup HTTPS for Local Wallet Pairing

The HashioDAO app utilizes the hashconnect library to pair with supported wallet extensions. Currently, the only supported wallet extension is HashPack. The HashConnect 1-click pairing feature only works in an SSL secured environment (https URLs). Enable HTTPS in your local build by creating a .env file in the root of this project and adding the HTTPS environment variable to it.

Add the HTTPS environment variable to your .env file and set it to true:

```
{% code title=".env" %}
```

```
HTTPS=true
```

```
{% endcode %}
```

Create an SSL certificate. There are several tools that can be used to generate a certificate and key. An easy way to do this is to use the mkcert tool.

Install mkcert via Homebrew (on macOS):

```
{% code overflow="wrap" %}
```

```
bash
```

The Homebrew macOS package manager is used for this example

```
Install mkcert tool
```

```
brew install mkcert
```

```
Install nss (only needed if you use Firefox)
```

```
brew install nss
```

```
Setup mkcert on your machine (creates a CA)
```

```
mkcert -install
```

```
{% endcode %}
```

Generate the certificate and key, storing them in a .cert directory:

```
bash
```

Create a directory to store the certificate and key

```
mkdir -p .cert
```

Generate the certificate (ran from the root of this project)

```
mkcert -key-file ./cert/key.pem -cert-file ./cert/cert.pem "localhost"
```

Set the SSLCRTFILE and SSLKEYFILE environment variables to the path of the certificate and key files. Add the variables to your .env file:

```
{% code title=".env" %}
```

```
/ Path to certificate /
```

```
SSLCRTFILE=./cert/cert.pem
```

```
/ Path to key /
```

```
SSLKEYFILE=./cert/key.pem
```

```
{% endcode %}
```

```
{% hint style="info" %}
```

Note: Make sure to include .env and .cert in your .gitignore file so this information is not committed to version control.

```
{% endhint %}
```

Setup Pinata Environment Variables

The HashioDAO app stores and retrieves IPFS data using Pinata. A Pinata public key, secret key, and gateway URL are necessary for IPFS pinning and fetching features to work as intended. If you have not already done so, create a Pinata account to generate a new set of keys and a gateway URL. [Link](#)

Add the below environment variables to your .env file to use the Pinata IPFS API:

```
{% code title=".env" %}
```

```
PRIVATEPINATAAPIKEY=/ Public Key /
```

```
PRIVATEPINATAAPISECRETKEY=/ Secret Key /
```

```
VITEPUBLICPINATAGATEWAYURL=/ Gateway URL /
```

```
{% endcode %}
```

A more comprehensive tutorial can be found in the [Pinata API Docs](#).

Step 3: Run Application

Run the application using the below command:

```
bash
```

```
vercel dev
```

This command will start your application, and you should see an https:// prefixed URL for your local server, indicating that HTTPS is successfully enabled.

Additional Resources

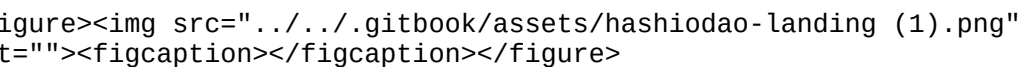
- [→ HashioDAO Repository](#)
- [→ Pinata API Documentation](#)
- [→ HashPack Documentation](#)

multisig-dao.md:

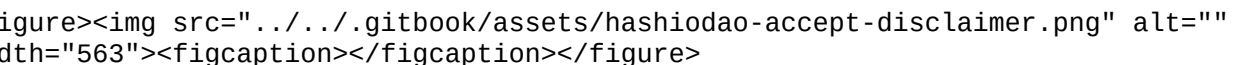
Multisig DAO

Multisig DAO Creation

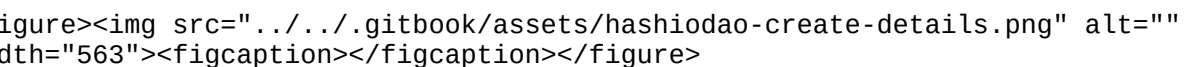
After creating and funding a Hedera account, navigate to the HashioDAO web application dashboard and connect your wallet. Once your wallet is connected, click Create a new DAO and this will guide you through a series of setup steps where you will define the details of your DAO including name and governance type.

The image shows the HashioDAO landing page, which includes a header with the HashioDAO logo and navigation links, and a main content area with a large heading and introductory text.

Read and accept the disclaimer and click Next.

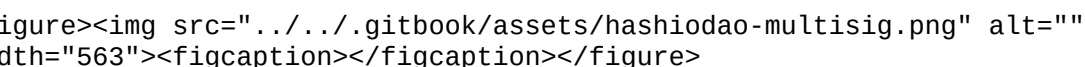
The image shows the HashioDAO accept disclaimer screen, which features a large heading, a paragraph of text, and a 'Next' button at the bottom right.

Define the details of your DAO by filling out each field. These details are important because it creates your DAO's public profile and how the community will be identified.

The image shows the HashioDAO create details screen, which contains several input fields for defining the DAO's details, such as name, token, and governance model.

Governance Models

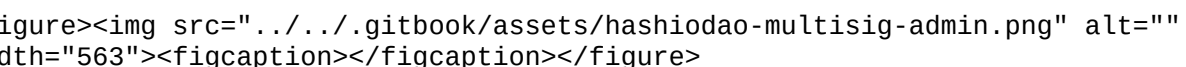
Choose the governance model that works best with your organizations's goals. There are three models to choose from: governance token, multisig, and NFT.

The image shows the HashioDAO multisig screen, which displays the 'Multisig' option as the selected governance model.

The createDAO() function is called on the MultisigDaoFactory factory contract when you create a multisig DAO. This initiates a new DAO treasury smart contract based on the parameters you set for your token. These parameters will be set in the next step.

Set Admin Account

Setting an admin account ID designates the account as the primary administrator responsible for initiating transactions and managing the multisig wallet.

The image shows the HashioDAO multisig admin screen, which includes a field for entering the admin account ID and a 'Next' button.

Define Threshold

Define the threshold. This is the minimum number of signers required before executing transactions.

<figure><figcaption></figcaption></figure>

Review and Submit to Create DAO

Review the DAO details, governance type, and submit the transaction. Approve the DAO creation transaction in your wallet to proceed.

<figure><figcaption></figcaption></figure>

Conclusion and Additional Resources

Congrats on successfully creating your first DAO using the multisig model! In this guide, you defined and customized the details of the new DAO, including specifying the admin account and threshold for authorizing transactions. Happy DAO creating!

- [➞ HashioDAO Repository](#)
- [➞ HashScan Network Explorer](#)
- [➞ HashioDAO Web Application](#)
- [➞ Gnosis Safe Governance Docs](#)

nft-dao.md:

NFT DAO

NFT DAO Creation

After creating and funding a Hedera account, navigate to the HashioDAO web application dashboard and connect your wallet. Once your wallet is connected, click Create a new DAO and this will guide you through a series of setup steps where you will define the details of your DAO including name and governance type.

<figure><figcaption></figcaption></figure>

Read and accept the disclaimer and click Next.

<figure><figcaption></figcaption></figure>

Define the details of your DAO by filling out each field. These details are important because it creates your DAO's public profile and how the community will be identified.

<figure><figcaption></figcaption></figure>

Governance Models

Choose the governance model that works best with your organizations's goals. There are three models to choose from: governance token, multisig, and NFT.

<figure><figcaption></figcaption></figure>

The createDAO() function is called on the NFTDaoFactory factory contract when you create an NFT DAO. This initiates a new DAO treasury smart contract based on the parameters you set for you token. These parameters will be set in the next step.

Define NFT and Voting Details

Define the DAO's NFT and voting details. You have the option to create a new governance token or use an existing one. For the purposes of this tutorial, we will create a new one. Define the details for your new governance token:

<figure><figcaption></figcaption></figure>

Once you click Create Token and approve the transaction in your wallet a Governor smart contract is deployed for managing proposals, voting, and executing decisions. This contract streamlines proposal handling and ensures a fair voting process based on community consensus. This transaction will also generate a token ID for your governance token.

<figure><figcaption></figcaption></figure>

Set the voting quorum percentage, voting duration, and delay period for votes. To avoid delays, starting with a low quorum percentage is recommended. This can be adjusted over time as the community's needs change.

{% hint style="info" %}

DAOs set their quorum percentage depending on their goals and needs. If the quorum is set too high, decisions are only made with a large number of participation from the members. If the quorum is set low, this allows more flexibility in decision making and the chances of proposals being approved would be higher than with a higher quorum.

{% endhint %}

<figure><figcaption></figcaption></figure>

Be cautious when depositing governance tokens into the treasury since it does not have voting rights. If you deposit a large portion it could lock you out of proposal creation, voting, and treasury access.

Review and Submit to Create New DAO

Review the DAO details, governance type, and submit the transaction. Approve the DAO creation transaction in your wallet to proceed.

<figure><figcaption></figcaption></figure>

Mint and Deposit NFT

Once your DAO is created, you need to mint the NFT and deposit to the treasury to finish setting up. Click the Mint NFT button, enter the NFT URL, and click the Mint NFT Tokens button.

<figure><figcaption></figcaption></figure>

{% hint style="info" %}

When a DAO is created, its financial resources are managed through the DAO treasury smart contract. This contract facilitates the distribution of funds by allowing transactions to be initiated by the admin. In this structure, the role of the admin is fulfilled by the voting (Governor) contract.


{% endhint %}

When the "Successfully minted NFT tokens" window pops up, click on the Deposit button and select the NFT and serial number from dropdown, and Deposit Fund.

<figure><figcaption></figcaption></figure>

Conclusion and Additional Resources

Congrats on successfully creating your first DAO using the NFT governance model! In this guide, you defined and customized the details of the new DAO, including minting governance NFTs. Happy DAO creating!

 [HashioDAO Repository](#)

 [HashScan Network Explorer](#)

README.md:

HashioDAO

Introduction

Distributed ledger technology (DLT) enables the shift from centralized control into the world of decentralization. Instead of having one central authority making all the decisions, decentralization gives the power and control back to the users. This enables new use cases in finance, governance, voting, and fundraising. In this documentation, you will learn about one use case: decentralized autonomous organizations (DAOs). By the end of this, you'll have a better understanding of what and how DAOs and HashioDAO function.

What is a DAO?

A DAO is a collective where members can make decisions, like managing financial assets or controlling a smart contract. These communities operate without a central authority, so decision-making powers are distributed among members who collectively make decisions through proposals and voting processes within a decentralized governance model. All transactions and decisions are auditable and transparent as they are recorded on a public ledger.

The key components of a DAO include smart contracts for automating proposals and managing funds, governance tokens that represent voting rights, and decentralized governance, where members collectively make decisions through proposals and voting. These components work together to create a decentralized, transparent, and community-driven organization that operates autonomously on a public network, like Hedera.

What is HashioDAO?

HashioDAO is a decentralized platform that offers a user-friendly interface for creating and managing DAOs. It provides customizable governance tokens, multisig capabilities, and treasury management tools. These features empower communities to define their governance model to best fit their preferences, simplifying voting and proposal management without the need for extensive technical knowledge. This opens up decentralized governance to a broader audience.

The platform offers three flexible governance models, so communities can choose the structure that fits their needs:

Governance Token: This model gives decision-making powers to token holders, making it perfect for DAOs that have large communities. It may be more vulnerable to governance attacks if tokens are owned by a small group.

NFT: This model gives decision-making powers to the holders of a specific NFT and allows more features than a regular token model like soulbound tokens or limiting the number of NFTs an account can own.

Multisig: This model requires two or more members to authorize and sign transactions. This structure helps reduce the risk of governance attacks by sharing authority among select members, making it more of a centralized governance model.

The platform offers multiple proposal templates to get members involved in shaping the DAO's future and making decisions:

Text Proposal: For presenting ideas or suggestions through written proposals.

Token Transfer Proposal: For requesting treasury funds for projects and payments.

Token Associate Proposal: For associating new tokens with the DAO's treasury.

Upgrade DAO Proposal: For implementing smart contract upgrades or enhancing security features.

HashioDAO: How It All Works Together

```
<figure><picture><source srcset="../../../gitbook/assets/hashiodao-contracts-system-dark-mode.png" media="(prefers-color-scheme: dark)"></picture><figcaption></figcaption></figure>
```

HashioDAO provides a comprehensive architecture that utilizes a system of smart contracts that work together to simplify DAO creation and management. The main components of the architecture and how it works:

1. **Factory Contracts:** The platform uses factory smart contracts (FTDaoFactory, NFTDaoFactory, or MultisigDaoFactory). A Factory smart contract is a design pattern used in decentralized applications (dApps) to allow efficient deployment of multiple smart contracts with similar functionalities. These factory contracts have been audited by CertiK.

2. **Treasury Contract:** When you call the `createDAO()` function, the factory contract creates the new DAO treasury smart contract based on the parameters you set for your token. This treasury contract holds and manages your DAO's funds,

including the governance tokens and any other tokens you associate with it. It also ensures all predefined rules set by the governance model are enforced, like quorum requirements and executing proposals.

3. Governor Contract: Along with the treasury contract, the factory contract also creates a separate Governor contract for each DAO. This Governor contract handles everything related to proposals, like creating, voting, and executing them. When a proposal is created, the Governor contract stores all the details, like the description and voting options, on InterPlanetary File System (IPFS) to keep things decentralized and secure.

So, once voting on a proposal ends, the Governor contract validates the results. If the proposal passes, the Governor contract tells the Treasury contract to execute the proposed actions, like transferring funds or associating new tokens. By using the factory pattern and IPFS, HashioDAO makes creating and managing DAOs efficient and decentralized.

Getting Started with HashioDAO

To get started with using HashioDAO, complete these prerequisites:





- Create a Hedera account.
- Fund the Hedera account.
- Download the HashPack wallet extension.

{% hint style="info" %}

📌 Note: The cost of creating a DAO is 1 HBAR. It is recommended that you test out the features of HashioDAO on Hedera Testnet first. This is where you can experiment and familiarize yourself with the platform before jumping into HashioDAO on Hedera Mainnet.

{% endhint %}

Once you have completed the above, choose the governance model and follow the steps to create your first:

 governance-token-dao.md	Governance Token DAO
 nft-dao.md	NFT DAO
 multisig-dao.md	Multisig DAO
 dao-proposals.md	DAO Proposals

how-to-use-it.md:

How to use it

Usage Guide

To install the hedera-custodians-library from NPM, run the following command in your terminal:

```
sh
npm install @hashgraph/hedera-custodians-integration
```

This command will install the hedera-custodians-library along with all its

necessary dependencies. Ensure you are in your project's root directory or the appropriate subdirectory where you wish to add the library.

Creating a Service

To create a custodial wallet service:

1. Choose the appropriate configuration (FireblocksConfig or DFNSConfig).
2. Instantiate the configuration with the required parameters.
3. Create a CustodialWalletService instance using the configuration.

```
typescript
import { CustodialWalletService, FireblocksConfig } from '@hashgraph/hedera-custodians-integration';

const config = new FireblocksConfig(
  FIREBLOCKSAPIKEY,
  FIREBLOCKSAPISECRETKEY,
  FIREBLOCKSBASEURL,
  FIREBLOCKSVAULTACCOUNTID,
  FIREBLOCKSASSETID
);

const service = new CustodialWalletService(config);
```

Signing Transactions

To sign a transaction:

1. Create a SignatureRequest with the transaction bytes.
2. Use the signTransaction method of the CustodialWalletService.

```
typescript
import { SignatureRequest } from '@hashgraph/hedera-custodians-integration';

const transactionBytes = new Uint8Array([/ your transaction bytes /]);
const request = new SignatureRequest(transactionBytes);

const signature = await service.signTransaction(request);
```

Managing Multiple Wallets

To manage multiple wallets:

1. Create separate configurations for each wallet.
2. Instantiate a CustodialWalletService for each configuration.

```
typescript
const fireblocksConfig = new FireblocksConfig(/ Fireblocks parameters /);
const dfnsConfig = new DFNSConfig(/ DFNS parameters /);

const fireblocksService = new CustodialWalletService(fireblocksConfig);
const dfnsService = new CustodialWalletService(dfnsConfig);
```

Additional Resources

 [Hedera Custodians Library Repository](#)

README.md:

Hedera Custodians Library

Introduction

The Hedera Custodians Library is a TypeScript utility designed to simplify significantly managing a custodial wallet and its associated accounts for integration with the Hedera network. It gives developers a strong base for managing custodial wallets within their TypeScript applications, allowing them to focus on core logic.

Key Features

- Simplifies integration with the Hedera network
- Abstracts away complexities of custodial wallet management
- Provides a unified interface for different custodial services
- Enhances security and compliance in digital asset management

By leveraging the Hedera Custodians Library, developers can efficiently implement secure and scalable custodial wallet solutions in their Hedera-based applications.

Custodial Wallet Management

Custodial wallet management is the practice of a third party being trusted with storage and security concerning the private keys that are associated with the cryptocurrency. Within the Hedera ecosystem, custodial wallet management is essential for the following reasons:

- Security: Custodial services have robust security and safety mechanisms to maintain digital assets.
- Institutional Adoption: Many institutional investors require custodial solutions to comply with regulatory requirements and internal risk management policies.
- Simplified User Experience: Custodial services may abstract the complexities of key management, making the user's interaction with the Hedera network much easier.
- Integration with Traditional Finance: This often bridges cryptocurrency and traditional finance, furthering adoption.
- Multi-signature Support: It can be done in various ways through custodial implementations with multi-signature wallets, which contribute to higher security and make more complex governance structures possible.
- Regulatory Compliance: Custodial services typically have some in-built compliance features to assist users in conforming to KYC/AML.

The Hedera Custodian Library simplifies the integration of custodial wallet services into applications built on the Hedera network, thus enabling developers to access the above features without integrating complex custodial logic. Currently, the library supports two leading custodial services:

Fireblocks

Fireblocks is a digital asset storage, transfer, and settlement platform specifically built for financial institutions and other businesses transacting in cryptocurrencies and digital assets. In the context of the Hedera ecosystem:

- Secure storage: HBAR is stored securely on Fireblocks alongside Hedera tokens, using a combination of MPC (Multi-Party Computation) technology and hardware isolation.
- Securing transactions: It offers a policy engine that enables the setup of custom approval flows for every transaction conducted on the Hedera network.
- Integration: Fireblocks can be integrated into the Hedera network, allowing seamless transactions with Hedera-based assets and their custodial management.

API access: The Hedera Custodian Library has full access to the API, which it uses for wallet automation and signing transactions.

DFNS

DFNS (Digital Financial Network & Security) is a custodial wallet infrastructure provider that offers MPC-based key management and transaction signing services. In the Hedera ecosystem:

Key Management: DFNS provides secure key generation and management for Hedera accounts without exposing private keys.

Programmable Authorization: It offers customizable authorization policies for Hedera transactions, enabling complex approval flows.

Multi-Tenancy: DFNS supports multi-tenant architectures, allowing businesses to manage multiple Hedera accounts for different users or purposes.

API-First Approach: The service provides RESTful APIs that the Hedera Custodian Library uses to interact with Hedera accounts and sign transactions.

Fireblocks and DFNS enable secure, scalable, and compliant management of Hedera accounts and assets. By supporting these services, the Hedera Custodian Library allows developers to choose the custodial solution that best fits their needs while providing a unified interface for interacting with custodial wallets in the Hedera ecosystem.

API Reference

<details>

<summary>CustodialWalletService</summary>

The CustodialWalletService class is the main entry point for interacting with custodial wallets.

Constructor

```
typescript
constructor(config: FireblocksConfig | DFNSConfig)
```

Creates a new instance of the CustodialWalletService with the specified configuration.

Methods

```
typescript
async signTransaction(request: SignatureRequest): Promise<Uint8Array>
```

Signs a transaction using the configured custodial service.

Parameters:

request: A SignatureRequest object containing the transaction to be signed.

Returns: A Promise that resolves to a Uint8Array containing the signature.

</details>

<details>

<summary>FireblocksConfig</summary>

The FireblocksConfig class represents the configuration for the Fireblocks custodial service.

Constructor

```
typescript
constructor(
  apiKey: string,
  apiSecretKey: string,
  baseUrl: string,
  vaultAccountId: string,
  assetId: string
)
```

Creates a new FireblocksConfig instance.

Properties

apiKey: The API key for Fireblocks.
 apiSecretKey: The API secret key for Fireblocks.
 baseUrl: The base URL for the Fireblocks API.
 vaultAccountId: The Fireblocks vault account ID.
 assetId: The asset ID for the Hedera token in Fireblocks.

</details>

<details>

<summary>DFNSConfig</summary>

The DFNSConfig class represents the configuration for the DFNS custodial service.

Constructor

```
typescript
constructor(
  serviceAccountAuthorizationToken: string,
  serviceAccountCredentialId: string,
  serviceAccountPrivateKey: string,
  appOrigin: string,
  appId: string,
  walletId: string
)
```

Creates a new DFNSConfig instance.

Properties

serviceAccountAuthorizationToken: The authorization token for the DFNS service account.
 serviceAccountCredentialId: The credential ID for the DFNS service account.
 serviceAccountPrivateKey: The private key for the DFNS service account.
 appOrigin: The origin URL of the DFNS app.
 appId: The ID of the DFNS app.
 walletId: The ID of the DFNS wallet.

</details>

<details>

<summary>SignatureRequest</summary>

The SignatureRequest class represents a request to sign a transaction.

Constructor

```
typescript
constructor(transactionBytes: Uint8Array)
```

Creates a new `SignatureRequest` instance.

Properties

`transactionBytes`: A `Uint8Array` containing the transaction bytes to be signed.

</details>

metamask-hedera-wallet-snap-tutorial.md:

description: >-

A step-by-step tutorial on how to integrate the Hedera Wallet Snap by MetaMask into a dApp.

Tutorial: MetaMask Snaps – What Are They and How to Use Them

Introduction

MetaMask is a widely used Ethereum wallet and browser extension – MetaMask Snaps is an open-source solution designed to enhance the capabilities of this wallet. Snaps are created by developers using JavaScript and enable users to interact with various blockchains, protocols, and decentralized applications (dApps) that MetaMask does not natively support. To learn more about Snaps, visit the MetaMask Snap Guide.

The Hedera Wallet Snap, developed by Tuum Tech and managed by Hashgraph, enables users to interact directly with the Hedera network. It offers functionalities like sending HBAR to different accounts and retrieving account information.

What You Will Learn

This tutorial will demonstrate how dApp builders and developers can seamlessly integrate and utilize the Hedera Wallet Snap in their applications. You will learn how to:

- Pair dApp with MetaMask

- Install the Hedera Wallet Snap

- Get the Snap Ethereum Virtual Machine (EVM) address

- Create the Snap account and check its balance

- Call other methods in the Hedera Wallet Snap, like: `transferCrypto`

Tools You Will Use

- [React JS \(Documentation\)](#)

- [MetaMask \(Documentation\)](#)

- [Hedera Wallet Snap for MetaMask \(Documentation\)](#)

- [Hedera JSON-RPC Relay \(Hashio\)](#)

- [Ethers JS \(Documentation\)](#)

- [Mirror Node REST API \(Learn More\)](#)

- [Mirror Node Explorer \(HashScan\)](#)

Prerequisites

- NodeJS `>= 18.13` ([Download](#))

TypeScript >= 4.7 (Download)
Git Command Line (Download)
Hedera Testnet Account (Create)
MetaMask Wallet Extension (Download)

Get Familiar with the dApp Structure and UI

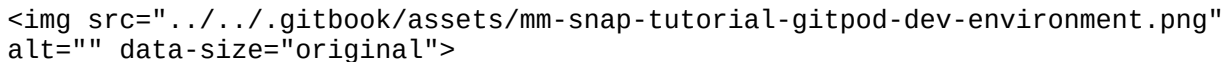
Explore the project files and functions and get a feel for how the sample dApp looks and functions. This will make it easier to follow along as you dive into the technical aspects of the dApp.

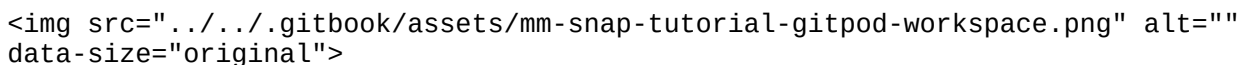
For convenience, two options are provided to run the code used in this example:

<details>

<summary>Run in Browser Using GitPod</summary>

This option does not require installing anything on your machine. All you need is a compatible web browser. Click [here](#) to set up the GitPod environment. You will see the following:

alt="" data-size="original">

alt="" data-size="original">

</details>

<details>

<summary>Clone the Repository Locally</summary>

If you prefer to have a copy of the code files on your machine and run the application locally, follow these steps.

Clone the Repo

To clone the repository, open your terminal and navigate to the directory where you want to place the project. Then, run the following command:

```
bash
git clone https://github.com/hedera-dev/hedera-example-metamask-snap.git
```

Navigate to Directory

Once the cloning process is complete, navigate to the project folder using:

```
bash
cd hedera-example-metamask-snaps
```

Install Project Dependencies and Start the Application

After cloning the repo and navigating to the right folder, install all project dependencies. Dependencies are listed in the package.json file, so you can just run the following command:

```
bash
```



```
npm install
```

To start the application, run:

```
bash
npm start
```

</details>

Project Structure

The project folder structure should look something like the following.

```
<div align="left">
```

```
<figure><figcaption></figcaption></figure>
```

```
</div>
```

Overall dApp Structure

The example application has four buttons, which complete different tasks when pressed.

- The first button connects the application to MetaMask.
- The second button installs the Hedera Wallet Snap.
- The third button obtains information about the Snap account.
- The fourth button (and respective input boxes) uses the Hedera Wallet Snap to transfer HBAR.

```
<figure><figcaption></figcaption></figure>
```

Now let's look at the App.jsx file (inside the src folder) behind this UI. You can think of the code as three main sections (in addition to the imports):

1. The state management is the part that uses the useState() React Hook.
2. The functions that are executed with each button press; we'll look at these in the next few sections.
3. The return statement groups the elements we see on the page.

```
<figure><figcaption></figcaption></figure>
```

The useState() hook helps store information about the state of the application. In this case, we store information like the snapId, wallet data, the account that is connected, the network, and the receiver address and HBAR amount, along with text and links presented in the UI. Remember that the first output of useState() is the variable of interest (e.g., walletData), and the second output is a function to set a new value for that variable (e.g., setWalletData).

Understanding the React Components

The buttons, text, and input boxes in the UI are a combination of React components in the return statement of App.jsx. Working with components, we take advantage of React's composability for better organization and readability. The properties for each component instance – like the function that each button executes, the label of the button, the text above the button, and the link for that text – are customized using React props.

<details>

<summary>MyGroup</summary>

```
jsx
import React from "react";
import MyButton from "../MyButton.jsx";
import MyText from "../MyText.jsx";
function MyGroup(props) {
  return (
    <div>
      <MyText text={props.text} link={props.link} />
      <MyButton fcn={props.fcn} buttonLabel={props.buttonLabel} />
    </div>
  );
}
export default MyGroup;
```

MyGroup is a functional component that combines a text element with a button and receives props as an argument. These properties include:

text: The text to be displayed by the MyText component.

link: An optional link to be associated with the MyText component. If provided, the text will become clickable and redirect to the specified link.

fcn: A function to be executed when the button within the MyButton component is clicked.

buttonLabel: The label for the button in the MyButton component.

Inside the component, we return a div element that contains both the MyText and MyButton components. We pass the corresponding props down to these child components, allowing them to render the text, link, and button label and assign the click event handler. Finally, by exporting MyGroup, we make it available for use in other parts of the application, enabling us to quickly create reusable groups of text and button elements throughout the dApp. You can find the JSX files for the functional components mentioned under the folder src/components.

</details>

<details>

<summary>MyButton</summary>

```
jsx
import React from "react";
function MyButton(props) {
  return (
    <div>
      <button onClick={props.fcn} className="cta-button">
        {props.buttonLabel}
      </button>
    </div>
  );
}
export default MyButton;
```

MyButton accepts the following props:

fcn: A function to be executed when the button is clicked.

buttonLabel: The label for the button, which will be displayed as the button's text.

Inside the component, we return a div element that wraps a button element. The button element is assigned the onClick event handler with the function

props.fcn. This allows us to execute a specified function when the button is clicked. The className attribute is set to "cta-button," which is used for styling the button with CSS. Finally, we display the props.buttonLabel as the button's text.

By exporting MyButton, we make it available for use in other groups or parts of the application, allowing us to easily create consistent and reusable buttons throughout the dApp with customized functionality and labels.

</details>

<details>

<summary>MyText</summary>

```
jsx
import React from "react";
function MyText(props) {
  if (props.link !== "") {
    return (
      <div>
        <a href={props.link} target={"blank"} rel="noreferrer">
          <p className="sub-text">{props.text}</p>
        </a>
      </div>
    );
  } else {
    return (
      <div>
        <p className="sub-text">{props.text}</p>
      </div>
    );
  }
}
export default MyText;
```

MyText displays text and optionally wraps it in a link. It takes props as an argument, including:

text: The text to be displayed.

link: An optional link to be associated with the text.

Inside the component, we use a conditional statement to check if props.link is provided. If it is, we wrap the text within an <a> element, making it clickable and redirecting to the specified link. If no link is provided, we display the text within a <p> element. Both elements have the className "sub-text" for consistent styling. Finally, we export the MyText component so it can be used in other parts of the application, allowing for easy creation of text elements with optional links.

</details>

<details>

<summary>MySendGroup</summary>

```
typescript
import React from "react";
import MyInputBox from "./MyInputBox";
import MyButton from "./MyButton";
import MyText from "./MyText";
function MySendGroup(props) {
  return (
```

```

        <div>
          <MyText text={props.textapp} link={props.linkapp} />
          <div className="multi-col-group">
            <MyInputBox fcn={props.fcnI1app}
placeholderTxt={props.placeholderTxt1app} />
            <MyInputBox fcn={props.fcnI2app}
placeholderTxt={props.placeholderTxt2app} />
            <MyButton fcn={props.fcnB1app}
buttonLabel={props.buttonLabelapp} />
          </div>
        </div>
      );
    }
    export default MySendGroup;

```

MySendGroup uses a text display, two input boxes, and a button to facilitate data entry and user interaction. The component takes props as an argument, including:

fcni1 and fcni2: Functions to store the text provided via input fields in the application state.

placeholderTxt1app and placeholderTxt2app: Placeholder texts for the input fields.

The same arguments are mentioned for MyText and MyButton.

Inside the component, we return a div element that contains the MyText, MyInputBox, and MyButton components. We pass the corresponding props down to these child components. Finally, exporting MySendGroup makes it available for use in other parts of the application.

</details>

<details>

<summary>MyInputBox</summary>

```

jsx
import React from "react";
function MyInputBox(props) {
  return (
    <div>
      <input type="text" onChange={props.fcn}
placeholder={props.placeholderTxt} className="text-input" />
    </div>
  );
}
export default MyInputBox;

```

MyInputBox accepts the following props:

fcn: A function to store the text provided via input fields in the application state.

placeholderTxt: Placeholder texts for the input fields.

Inside the component, we return a div that wraps an input element. The input element is assigned the onChange event handler with the function props.fcn. This triggers the function that stores the inputs (receiver address and HBAR amount) to the state of the application. We display the props.placeholderTxt as the box text. The className attribute is set to "text-input," which is used for styling the boxes with CSS. Finally, by exporting MyInputBox, we make it available for use in other parts of the application.

</details>

Step 1: Pair MetaMask Wallet (select network and account)

In App.jsx, we use the connectWallet() function (code tab 1), which in turn calls the walletConnectFcn() function (code tab 2) that is imported from the file src/components/hedera/walletConnect.js.

```
{% tabs %}
{% tab title="connectWallet()" %}
jsx
async function connectWallet() {
  if (account !== undefined) {
    setConnectText( Account ${account} already connected ⚡ ☒);
  } else {
    const wData = await walletConnectFcn(network);

    let newAccount = wData[0];
    if (newAccount !== undefined) {
      setConnectText( Account ${newAccount} connected ⚡ ☒);
      setConnectLink(https://hashscan.io/${network}/account/${newAccount});
      setWalletData(wData);
      setAccount(newAccount);
      setSnapInstallText();
      setSnapInfoText();
      setSnapTransferText();
    }
  }
}

{% endtab %}

{% tab title="walletConnectFcn()" %}
jsx
import { ethers } from "ethers";
async function walletConnectFcn(network) {
  console.log(\n=====);
  // ETHERS PROVIDER
  const provider = new ethers.providers.Web3Provider(window.ethereum, "any");
  // SWITCH TO HEDERA TEST NETWORK
  console.log(- Switching network to the Hedera ${network}...);
  let chainId;
  if (network === "testnet") {
    chainId = "0x128";
  } else if (network === "previewnet") {
    chainId = "0x129";
  } else {
    chainId = "0x127";
  }
  await window.ethereum.request({
    method: "walletaddEthereumChain",
    params: [
      {
        chainName: Hedera ${network},
        chainId: chainId,
        nativeCurrency: { name: "HBAR", symbol: "HBAR", decimals: 18 },
        rpcUrls: [https://${network}.hashio.io/api],
        blockExplorerUrls: [https://hashscan.io/${network}/],
      },
    ],
  },
```

```

});
console.log("- Switched ✓");
// // CONNECT TO ACCOUNT
console.log("- Connecting wallet...");
let selectedAccount;
await provider
  .send("ethrequestAccounts", [])
  .then((accounts) => {
    selectedAccount = accounts[0];
    console.log(- Selected account: ${selectedAccount} ✓);
  })
  .catch((connectError) => {
    console.log(- ${connectError.message.toString()});
    return;
  });
return [selectedAccount, provider];
}
export default walletConnectFcn;

{% endtab %}
{% endtabs %}

```

When the `<mark style="background-color:purple;">Connect Wallet</mark>` button is pressed in the dApp, the `connectWallet()` function is executed. This function checks if an account is already connected. If it is, a message displaying the connected account is shown. If no account is connected, the `walletConnectFcn()` is called to establish a connection.

The `walletConnectFcn()` function performs the following steps:

1. Creates an ethers provider: It initializes an ethers provider using the `Web3Provider` from the ethers library, which connects to MetaMask. An Ethers provider serves as a bridge between your application and the Hedera network. It allows you to perform actions like sending transactions and querying data.
2. Switches to Hedera Testnet: It determines the `chainId` based on the chosen Hedera network (testnet, previewnet, or mainnet) and sends a `wallet\addEthereumChain` request to MetaMask to add the corresponding Hedera network. A chain ID is a unique identifier that represents a blockchain network. This is an important step that includes setting the native currency (HBAR) and providing the JSON-RPC and network explorer URLs. For the JSON-RPC provider, this example uses Hashio, a community-hosted JSON-RPC relay provided by Hashgraph (note that anyone can host their own relay and/or use other commercial providers, like Arkhia). For network explorer, HashScan is used. (Keep in mind that HashScan supports EIP-3091, which makes it easy to explore historical data like blocks, transactions, accounts, contracts, and tokens from wallets like MetaMask).
3. Connects and Pairs Account: The function sends an `eth\requestAccounts` request to MetaMask to access the user's Hedera account. Upon successful connection, the selected account is returned.

Finally, the `connectWallet()` function in `App.jsx` updates the React state with the connected testnet account and provider information, allowing the dApp to display the connected testnet account information.

This is what you see when clicking the `<mark style="background-color:purple;">Connect Wallet</mark>` button for the first time.

```

{% hint style="info" %}
Note: The Hedera account selected in MetaMask for this example has ECDSA keys
and is created using the Hedera Portal.
{% endhint %}

```

```

<figure><figcaption></figcaption></figure>

```

Once the network switches and the account is paired, you should see something like the following in the dApp UI and in HashScan (if you click on the hyperlinked text showing the account address).

```
<figure><figcaption></figcaption></figure>
```

```
<figure><figcaption></figcaption></figure>
```

Step 2: Install the Hedera Wallet Snap

The `snapInstall()` function (code tab 1) in `App.jsx` calls the `snapInstallFcn()` function (code tab 2). The latter is imported from the file `src/components/hedera/snapInstall.js`.

```
{% tabs %}
{% tab title="snapInstall()" %}
jsx
async function snapInstall() {
  if (account === undefined) {
    setSnapInstallText("🔗 Connect a wallet first! 🔗");
  } else {
    const newSnapInstallText = await snapInstallFcn(snapId);
    setSnapInstallText(newSnapInstallText);
    setSnapInfoText();
  }
}

{% endtab %}

{% tab title="snapInstallFcn()" %}
jsx
async function snapInstallFcn(snapId) {
  console.log(`\n=====`);
  console.log(`- Installing Hedera Wallet Snap...`);
  console.log(`SnapId: ${snapId}`);
  let outText;
  let snaps = await window.ethereum.request({
    method: "walletGetSnaps",
  });
  console.log("Installed snaps...", snaps);
  try {
    if (!(snapId in snaps)) {
      console.log("Hedera Wallet Snap is not yet installed. Installing now...");
      const result = await window.ethereum.request({
        method: "walletrequestSnaps",
        params: {
          [snapId]: {},
        },
      });
      console.log("result: ", result);
      snaps = await window.ethereum.request({
        method: "walletGetSnaps",
      });
    }
  } catch (e) {
    console.log(`Failed to obtain installed snap: ${JSON.stringify(e, null, 4)}`);
    alert(`Failed to obtain installed snap: ${JSON.stringify(e, null, 4)}`);
  }
  if (snapId in snaps) {
    outText = "Snap installed ✅";
  }
}
```

```

        console.log(- Snap installed successfully ✅);
        alert("Snap installed successfully!");
    } else {
        console.log("Could not connect successfully. Please try again!");
        alert("Could not connect successfully. Please try again!");
    }
    return outText;
}
export default snapInstallFcn;

{% endtab %}
{% endtabs %}

```

The snapInstallFcn() function performs the following steps:

1. Gets Installed Snaps: The function requests a list of installed Snaps in the user's MetaMask wallet using the wallet\getSnaps method. It logs the currently installed Snaps.
2. Installs Snaps: The function checks if the Hedera Wallet Snap (identified by snapId) is already installed. If it's not installed, the function attempts to install it by calling wallet\requestSnaps and passing the snapId. It then logs the result of this installation attempt. After attempting installation, it checks again for installed Snaps.
3. Handles Installation Outcome: If the Snap is successfully installed (i.e., snapId is found in the list of installed snaps), it logs and alerts the user that the Snap installation was successful. If the installation fails (i.e., snapId is not found in the list), it logs and alerts the user that the connection could not be established successfully.
4. Returns Result: Lastly, the function returns a text message indicating whether the Snap was installed successfully.

The text output from snapInstallFcn() is used by snapInstall() in the front end of the application to update the message seen by the user.

This is what you see when clicking the `<mark style="background-color:purple;">Install Snap</mark>` button for the first time.

`<figure><figcaption></figcaption></figure>`

`<figure><figcaption></figcaption></figure>`

Step 3: Get the Snap EVM Address

Before we can start using the functionality of the Hedera Wallet Snap, this step obtains the Snap EVM address. In the next step, we send HBAR to that Snap EVM address to create the corresponding Snap account.

These steps are necessary because Snaps provide a way to create a secure and unique wallet associated with a user's MetaMask account without directly accessing the private key of the account. This separate Snap account is unique to the user's MetaMask account and remains accessible across different browsers or devices as long as the secret recovery phrase remains the same. For more information, read this section of the Hedera Wallet Snap documentation.

The snapGetAccountInfo() function (code tab 1) in App.jsx calls the snapGetAccountInfoFcn() function (code tab 2). The latter is imported from the file src/components/hedera/ snapGetAccountInfo.js.

```

{% tabs %}
{% tab title="snapGetAccountInfo()" %}

```



```

jsx
async function snapGetAccountInfo() {
  if (account === undefined || snapInstallText === undefined) {
    setSnapInfoText("🔗Connect a wallet and install the snap first!
    ");
  } else {
    const [snapAccountAddress, infoText] = await
snapGetAccountInfoFcn(network, walletData, snapId);
    setSnapInfoText(infoText);
    setInfoLink(https://hashscan.io/${network}/address/${
snapAccountAddress});
    setSnapTransferText();
  }
}

{% endtab %}

{% tab title="snapGetAccountInfoFcn()" %}
jsx
async function snapGetAccountInfoFcn(network, walletData, snapId) {
  console.log(\n=====);
  console.log(- Invoking GetAccountInfo...);
  let outText;
  let snapAccountEvmAddress;
  let snapAccountBalance;
  try {
    const response = await window.ethereum.request({
      method: "wallet_invokeSnap",
      params: {
        snapId,
        request: {
          method: "getAccountInfo",
          params: {
            network: network,
            mirrorNodeUrl: https://${network}.mirrornode.hedera.com,
          },
        },
      },
    });
    snapAccountEvmAddress = response.accountInfo.evmAddress;
    snapAccountBalance = response.accountInfo.balance.hbars;
    outText = Snap Account ${snapAccountEvmAddress} has $
{snapAccountBalance} hbar ✓;
  } catch (e) {
    snapAccountEvmAddress = e.message.match(/0x[a-fA-F0-9]{40}/)[0];
    outText = Go to MetaMask and transfer HBAR to the snap address to
activate it: ${snapAccountEvmAddress} 🔗;
  }
  console.log(- ${outText});
  console.log(- Got account info ✓);
  return [snapAccountEvmAddress, outText];
}
export default snapGetAccountInfoFcn;

{% endtab %}
{% endtabs %}

```

The snapGetAccountInfoFcn() function performs the following steps:

1. Requests Account Information: The function makes a request to MetaMask using the wallet\invokeSnap method. It sends the snapId and a request for the getAccountInfo method, along with the network details and the URL of the Hedera mirror node. For more details on this method, read this section of the Hedera Wallet Snap documentation.

2. Processes the Response: If the request is successful, the function extracts the EVM address of the Snap account and its balance in HBAR from the response. It then constructs a message stating the account's EVM address and balance.
3. Handles Errors: If there's an error (like the account not being created yet), the function extracts the EVM address from the error message. It then creates a message instructing the user to transfer HBAR to this address to create the Snap account.
4. Returns Results: The function logs the outcome message (either the account details or the activation instruction). It then returns the EVM address of the Snap account and the outcome message.

The outputs from `snapGetAccountInfoFcn()` are used by `snapGetAccountInfo()` in the front end of the application to update the message and link seen by the user.

This is what you see when clicking the `<mark style="background-color:purple;">Get Snap Account Info</mark>` button for the first time.

```
{% hint style="info" %}
```

Note: When invoking a Snap method for the first time, MetaMask checks for confirmation before connecting the Hedera Wallet Snap to the account.

```
{% endhint %}
```

```
<figure><figcaption></figcaption></figure>
```

Once the Snap is connected to the MetaMask account, you should see something like the following in the dApp UI and in HashScan (if you click on the hyperlinked text showing the Snap account address).

```
<figure><figcaption></figcaption></figure>
```

```
<figure><figcaption></figcaption></figure>
```

Step 4: Create Snap Account and Check Its Balance

We now know the EVM address for the Snap account. However, this Snap account has not yet been created on the Hedera network (testnet in this case). It's time to send HBAR to that address to create the actual Hedera account for the snap.

Perform a transfer of 10 HBAR to the Snap address. In this case, the Snap address is: `0xa7deaf8acd6be555740f9672dff34f510480f0c9`

```
{% hint style="info" %}
```

Note: The Hedera account selected in MetaMask for this example has ECDSA keys and is created using the Hedera Developer Portal.

```
{% endhint %}
```

```
<figure><figcaption></figcaption></figure>
```

Once the transfer is complete, check the balance of the Snap account by clicking on the `<mark style="background-color:purple;">Get Snap Account Info</mark>` button again. You should see something like the following in the dApp UI and in HashScan (if you click on the hyperlinked text showing the snap account address and its balance).

```
<figure><figcaption></figcaption></figure>
```

```
<figure><figcaption></figcaption></figure>
```

Step 5: Call Other Methods in the Hedera Wallet Snap

The last step in this exercise is to try other methods available in the Hedera Wallet Snap. As of version 0.1.3 of the Snap, the `transferCrypto` method enables transferring HBAR to other Hedera accounts from MetaMask. In this case, we send 0.2 HBAR from the Snap account to the MetaMask account. Let's see how this is done in the code.




The `snapTransferHbar()` function (code tab 1) in `App.jsx` calls the `snapTransferHbarFcn()` function (code tab 2). The latter is imported from the file `src/components/hedera/snapTransferHbar.js`.

```
{% tabs %}
{% tab title="snapTransferHbar()" %}
jsx
async function snapTransferHbar() {
  if (account === undefined || snapInstallText === undefined ||
snapInfoText === undefined) {
    setSnapTransferText("🔴Complete all the steps above first!🔴");
  } else {
    setSnapTransferText(Transferring...);
    const transferText = await snapTransferHbarFcn(network, walletData,
snapId, [receiverAddress, hbarAmount]);
    setSnapTransferText(`${transferText}`);
  }
}

{% endtab %}

{% tab title="snapTransferHbarFcn()" %}
jsx
async function snapTransferHbarFcn(network, walletData, snapId, args) {
  console.log(\n=====);
  console.log(- Invoking transferCrypto...);
  let outText;
  const receiverAddress = args[0];
  const hbarAmount = parseFloat(args[1]);
  const maxFee = 0.05;
  const transfers = [
    {
      asset: "HBAR",
      to: receiverAddress,
      amount: hbarAmount, // in Hbar
    },
  ];
  // If you're sending to an exchange account,
  // you will likely need to fill this out
  const memo = "";
  try {
    await window.ethereum.request({
      method: "wallet_invokeSnap",
      params: {
        snapId,
        request: {
          method: "transferCrypto",
          params: {
            network: network,
            transfers,
            memo,
```

```

        maxFee: maxFee,
      },
    },
  });
  outText = Transfer successful  | Get the snap account info again to
see the updated balance!;
  console.log(- `${outText}`);
} catch (e) {
  outText = Transaction failed. Try again ;
  console.log(- Transfer failed : ${JSON.stringify(e, null, 4)});
}
return outText;
}
export default snapTransferHbarFcn;

{% endtab %}
{% endtabs %}

```

The `snapTransferHbarFcn()` function performs the following steps:

Sets Transfer Details: It extracts the recipient's address and the amount of HBAR to be transferred from the arguments (`args`) provided. Note that these inputs are specified in the dApp UI – remember that the receiver address is that of the MetaMask account and the transfer amount is 0.2 HBAR. The function sets a maximum fee for the transaction, which is predefined as 0.05 HBAR in the code. It also prepares the transfer details, including the asset type (HBAR), the receiver's address, and the amount.

Executes Transfer Request: The function sends a `wallet\invokeSnap` request. This request includes the `snapId`, the `transferCrypto` method, and parameters such as the network, the transfer details, a memo (may be needed for transferring to exchange accounts), and the maximum fee.

Handles Transfer Outcome: If the transfer is successful, the function sets a message indicating success and suggests re-checking the Snap account info for an updated balance. In case of an error or failure, it sets a different message indicating that the transaction failed and prompts trying again.

Returns Output: Finally, the function returns the outcome message, informing whether the transfer was successful or not.

The text output from `snapTransferHbarFcn()` is used by `snapTransferHbar()` in the front end of the application to update the message seen by the user.

This is what you see when clicking the `<mark style="background-color:purple;">Transfer HBAR w/ Snap</mark>` button after entering a valid receiver address and HBAR amount in the corresponding input fields.

`<figure><figcaption></figcaption></figure>`

`<figure><figcaption></figcaption></figure>`

Once the transfer of HBAR from the Snap account to the MetaMask account is complete, you can check the Snap account balance by clicking on the `<mark style="background-color:purple;">Get Snap Account Info</mark>` button again. You will see something like this:

`<figure><figcaption></figcaption></figure>`

Summary

This tutorial provides a comprehensive guide on how to use MetaMask Snaps, focusing on the Hedera Wallet Snap.

It starts by covering the dApp's structure and user interface, highlighting the functions of various buttons like connecting to MetaMask, installing the Hedera Wallet Snap, obtaining Snap account information, and transferring HBAR using the Snap.

The tutorial also dives into the technical aspects, detailing the App.jsx file, which includes state management, functions for button actions, and the layout of the UI components. It explains the use of React components like MyGroup, MyButton, and MyInputBox to create an interactive and user-friendly interface.

It then elaborates on MetaMask Snaps, its purpose, and how to use them, specifically focusing on the Hedera Wallet Snap for tasks like pairing the dApp with MetaMask, installing the Snap, managing the Snap account, and executing crypto transfers.

🎉 Congratulations! You have learned about the new Hedera Wallet Snap by MetaMask and how to integrate it into a dApp. Feel free to reach out in Discord if you have any questions!

Additional Resources

- ➞ Project Repository
- ➞ Hedera Wallet Snap
- ➞ MetaMask Snap Guide
- ➞ Hedera JSON-RPC Relay Repository
- ➞ Ask Questions on StackOverflow

<p>Writer: Ed, Head of Developer Relations</p> <p>GitHub LinkedIn</p>	<p>Editor: Krystal, Technical Writer</p> <p>GitHub Hashnode</p> <p>https://github.com/theekrystallee</p>

README.md:

Hedera Wallet Snap By MetaMask



Overview

MetaMask is a popular Ethereum wallet and browser extension that developers can integrate into a variety of third-party applications. MetaMask Snaps is an open-source solution to enhance MetaMask's functionalities beyond its native capabilities. The Hedera Wallet Snap, developed by Tuum Tech and managed by Hashgraph, enables users to interact directly with the Hedera network without relying on Hedera JSON-RPC Relay, offering Hedera native functionalities like sending HBAR to different accounts and retrieving account information.

What is a Snap?

MetaMask Snaps is an open-source framework allowing secure extensions to MetaMask, thus enhancing web3 user experiences. It empowers the addition of new API methods, supports various blockchain protocols, and tweaks existing functionalities via the Snaps JSON-RPC API

Snaps enable users to interact with new blockchains, protocols, and decentralized applications (dApps) beyond what is natively supported by MetaMask. The goal of the MetaMask Snaps system is to create a more open, customizable, and extensible wallet experience for users while fostering innovation and collaboration within the blockchain and decentralized application ecosystem.

<p> Hedera Wallet Snap Documentation</p> <p>by Tuum Tech</p>	<p>https://docs.tuum.tech/hedera-wallet-snap/basics/introduction</p>
<p> Hedera Wallet Snap Tutorial</p> <p>https://docs.hedera.com/hedera/tutorials/smart-contracts/metamask-hedera-wallet-snap-tutorial</p>	

FAQs

<details>

<summary>What are the limitations for connecting MetaMask to the RPC vs the Snap?</summary>

The Hedera JSON RPC Relay supports only the methods defined at Hedera JSON RPC Relay Methods, which are limited to Hedera Smart Contract Services. In contrast, the Hedera Wallet Snap uses the Hedera SDK to interact natively with the ledger, allowing the future support of a wider range of Hedera services like Hedera Token Service, Hedera Consensus Service, and Hedera File Service, beyond just smart contracts.

</details>

<details>

<summary>How can I deploy a smart contract on Hedera using MetaMask?</summary>

To deploy a smart contract on Hedera using MetaMask, you will need to use the Hedera JSON RPC relay. You can deploy using tools compatible with EVM-based chains. For detailed steps, refer to Deploying Smart Contracts on Hedera.

</details>

<details>

<summary>Can a signer created via ED25519 account be used for Ethereum-based transactions?</summary>

No, you cannot use a signer created via ED25519 for Ethereum-based transactions due to the difference in cryptographic algorithms and key formats. EVM uses

ECDSA with the secp256k1 curve, which is different from ED25519. For interacting directly with smart contracts on Hedera, only ECDSA-based accounts can be used.

</details>

<details>

<summary>How can I delegate the signing process to MetaMask or WalletConnect when using Hedera SDK?</summary>

Currently, there is no direct way to delegate the signing process to MetaMask or WalletConnect for transactions composed by the Hedera SDK, as they do not provide private keys of users.

</details>

<details>

<summary>Can I use JSON RPC relay for transactions against Hedera native services?</summary>

The Hedera JSON RPC relay exposes specific methods, as detailed in Hedera JSON RPC Relay Methods. You can use these methods for transactions with Hedera's smart contracts. The Hedera Wallet Snap, using the Hedera SDK, can perform all Hedera transactions and will eventually support interactions with smart contracts as well.

</details>

<details>

<summary>Why should I use the Snap instead of adding the Hashio RPC info to MetaMask?</summary>

While Hashio RPC and other RPCs are limited to methods exposed by the Hedera JSON RPC relay, the Hedera Wallet Snap, using the Hedera SDK natively, offers access to all Hedera native features, including Hedera Token Service, Hedera File Service, and Hedera Consensus Service, enabling a broader range of interactions beyond smart contracts.

</details>

<details>

<summary>Is the Snap more suited for custom Hedera functionalities like HCS and HFS, which can't be done solely with MetaMask?</summary>

Yes, that's correct. The Hedera Wallet Snap is ideal for custom Hedera functionalities. It uses the Hedera SDK for all operations, allowing for native interactions with the full spectrum of Hedera's offerings.

</details>

pyth-network-oracle.md:

Pyth Oracles

What is Pyth?

The Pyth Network is a first-party financial oracle network designed to provide low-latency real-world data to multiple blockchains securely and transparently. Pyth currently supports 400+ real-time price feeds across crypto, equities, ETFs, FX pairs, and commodities and has facilitated more than \$100B in total

trading volume across over 50 blockchain ecosystems.

How to integrate with the Pyth Network on Hedera?

The Pyth Price Feeds uses a "pull" price update model, where users are responsible for posting price updates on-chain when needed. In the pull model, developers should integrate Pyth into both their on-chain and off-chain code:

1. On-chain programs should read prices from the Pyth program deployed on the same chain
2. Off-chain frontends and jobs should include Pyth price updates alongside (or within) their application-specific transactions.

Please note that it is possible to replicate the legacy Oracle design with the Pyth scheduler (previously known as "price pusher"). It is an off-chain application that regularly pulls price updates on to a blockchain you can find [here](#).

Demonstration: How to integrate with the Pyth Network on Hedera?

This demo is a simple example of using Pyth prices in Hedera and is based on the tutorial [here](#). Follow the instructions in the tutorial to build, deploy, and use this demo.

This demo is similar to the contract written in the tutorial. The only difference is that it uses a different math to calculate the price of 1\$ in HBAR because, In the Hedera EVM layer, the native token has only 8 decimal places, while Ethereum has 18 decimal places. This means that the smallest unit of HBAR (1 wei in Hedera EVM) is 0.00000001 HBAR, while the smallest unit of ETH is 0.000000000000000001 ETH. You can see the change in the code in the `MyFirstPythContract.sol`.

```
{% embed url="https://youtu.be/2ry0sTvnGo" %}
```

For more details, please visit this [GitHub repository](#).

If you have any questions, please refer to the [Pyth Network documentation](#) or join [Discord](#).

Contributors: @KemarTiti

README.md:

```
---
description: Oracle networks integrated with the Hedera network.
---
```

Oracle Networks

Oracle networks integrated with Hedera provide secure and reliable data feeds from off-chain sources to Hedera-based applications and smart contracts. These oracle networks act as bridges between the real world and the Hedera network, enabling the development of decentralized applications (dApps) that rely on external data.

Check out some of the oracle networks integrated with Hedera listed below. Each Oracle network may have its own unique features and data offerings.

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-cover data-type="files"></th><th
data-hidden data-card-target
```



```

data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><a
href="pyth-network-oracle.md"><strong>PYTH ORACLES</strong></a></td><td><a
href="../../.gitbook/assets/pyth-oracle-logo.png">pyth-oracle-logo.png</a></
td><td><a
href="pyth-network-oracle.md">pyth-network-oracle.md</a></td></tr><tr><td
align="center"><a href="supra-oracles.md"><strong>SUPRA
ORACLES</strong></a></td><td><a href="../../.gitbook/assets/supra-
oracles7134.jpg">supra-oracles7134.jpg</a></td><td><a href="supra-
oracles.md">supra-oracles.md</a></td></tr></tbody></table>

```

supra-oracles.md:

Supra Oracles

Overview

Supra is a novel, high-throughput Oracle & IntraLayer offering a vertically integrated toolkit of cross-chain solutions. These solutions include data oracles, asset bridges, and automation networks that aim to interlink all public and private blockchains. Supra provides decentralized Oracle price feeds to deliver real-world data to web3 ecosystems through various on-chain and off-chain use cases. Oracles ensure that the data from the real world is accurate, which is crucial for decentralized applications (dApps) that rely on real-world, real-time data. This is important for dApps that need real-time information, such as the prices of cryptocurrencies and various other assets. The Supra x Hedera integration aims to bring speed, security, and accuracy to real-time data feeds, enhancing the functionality and reliability of dApps on the Hedera network.

Developer Considerations

Contract calls, such as ethcall and ethestimateGas, go to the mirror node, which limits those to a small data payload size. Our engineering team has successfully upgraded the API call data payload capacity to 24 KB. This enhancement is designed to fetch a single price pair from the data feed efficiently, ensuring a more streamlined retrieval process.

{% hint style="info" %}

Please Note: While this update offers improved performance to pull single price pairs, attempting to pull more than one price pair at a time may surpass the new 24 KB data payload limit. Should this limit be exceeded, the API will return an error message. We recommend structuring your API calls accordingly to avoid any potential disruptions.

{% endhint %}

```

<table data-card-size="large" data-view="cards" data-full-
width="false"><thead><tr><th align="center"></th><th data-hidden data-card-cover
data-type="files"></th><th data-hidden data-card-target data-type="content-
ref"></th></tr></thead><tbody><tr><td align="center"><strong><img alt="Supra logo" data-bbox="715 728 735 740"/></strong> <a
href="https://supra.com/docs/readme/"><strong>Official
Documentation</strong></a></td><td><a href="../../.gitbook/assets/supra-
oracles7134.jpg">supra-oracles7134.jpg</a></td><td><a
href="https://supra.com/docs/readme/">https://supra.com/docs/readme/</a></td></
tr><tr><td align="center"><strong><img alt="Hedera logo" data-bbox="745 795 765 807"/></strong> <a
href="https://supra.com/data/catalog/details?
instrumentName=hbarusdt&#x26;providerName=supra"><strong>HBAR Price
Feed</strong></a></td><td><a href="../../.gitbook/assets/hedera-logo-black
(1).png">hedera-logo-black (1).png</a></td><td><a
href="https://supra.com/data/catalog/details?
instrumentName=hbarusdt&#x26;providerName=supra">https://supra.com/data/
catalog/details?instrumentName=hbarusdt&#x26;providerName=supra</a></td></tr></
tbody></table>

```

cli-management.md:

CLI Management

The Command Line Interface (CLI) is a core component of Stablecoin Studio and essential for developers aiming to streamline stablecoin management. This guide takes you from installing the SDK and CLI to customizing your config file and initiating stablecoin creation via CLI commands. Whether you're just getting started or already familiar with stablecoin management, this documentation provides the resources to navigate and optimize your stablecoin operations effectively.

Table of Contents

1. Prerequisites
2. Install Stablecoin Studio
3. Configure CLI
 1. Factory Contracts
 2. Deploy Factory Contracts
4. CLI Flow
5. Start CLI
6. Create Stablecoin
7. Operate Stablecoin
8. Configure Proof-of-Reserve
9. Additional Resources

Prerequisites

NodeJS >= 18.13
Solidity >= 0.8.16
TypeScript >= 4.7
Git Command Line
Hedera Testnet Account
HashPack Wallet

Install Stablecoin Studio

Open a new terminal and navigate to your preferred directory where you want your Stablecoin Studio project to live. Clone the repo using this command:

```
{% code fullWidth="false" %}  
bash  
git clone https://github.com/hashgraph/stablecoin-studio.git  
  
{% endcode %}
```

Install the npm package globally:

```
bash  
npm install -g @hashgraph/stablecoin-npm-cli
```

```
{% embed url="https://youtu.be/eThs08nUsLI?si=52IXyl74R8rilVr" %}  
How to Issue Stablecoins on Hedera: Studio Intro & Installation\  
by Ed Marquez  
{% endembed %}
```

Configure CLI

To use the CLI correctly, it is necessary to generate a configuration file where the default network, their associated accounts, and the factory contract ID will be included. These parameters can be modified later on.

Create a config file

From the root of the cli directory, start the command line tool and create a configuration file using the wizard command:

```
bash
cd stablecoin-studio/cli
accelerator wizard
```

The first time you execute the wizard command in your terminal, if you haven't added your default configuration path, the interface will prompt you to "Write your config path." To use the default configuration path, hit enter. This will walk you through the prompts where you will input your configuration settings and create your hsca-config-yaml file.

Let's go over the configuration details:

defaultNetwork

This sets the default network the application will connect to when it starts. It's essential for defining the environment where transactions will occur (e.g., testnet for testing, mainnet for production). We will be using testnet for this tutorial.

```
yaml
defaultNetwork: testnet
```

networks

This property contains a list of Hedera networks that the application can connect to. Each entry specifies the name of a network and a list of consensus nodes, allowing you to switch between different Hedera environments easily.

```
yaml
networks:
  - name: mainnet
    consensusNodes: []
  - name: previewnet
    consensusNodes: []
  - name: testnet
    consensusNodes: []
```

accounts

This property holds the credentials for various Hedera accounts. Each account has an accountId, a privateKey, and a network association. This is critical for performing transactions, as the private key is used to sign them. The alias field provides a user-friendly identifier for the account and importedTokens can store any tokens imported into this account. You can use the Hedera Developer Portal to create the default testnet account.

```
yaml
accounts:
```

- accountId: YOUR ACCOUNT ID
- privateKey:
 - key: >-
 - YOUR PRIVATE KEY
 - type: ED25519
- network: testnet
- alias: main
- importedTokens: []

mirrors

This property lists the available Hedera mirror nodes for each network. Mirror nodes hold historical data and can be queried for transactions, records, etc. The selected field indicates whether the mirror node is the default one to be used.

yaml

```
mirrors:
  - name: HEDERA
    network: testnet
    baseUrl: https://testnet.mirrornode.hedera.com/api/v1
    selected: true
  - name: HEDERA
    network: previewnet
    baseUrl: https://previewnet.mirrornode.hedera.com/api/v1
    selected: true
  - name: HEDERA
    network: mainnet
    baseUrl: https://mainnet.mirrornode.hedera.com/api/v1
    selected: true
```

rpcs

This property specifies the RPC (Remote Procedure Call) servers available for connecting to Hedera services. RPCs are essential for querying smart contracts, among other functionalities. Similar to mirror nodes, the selected field indicates the default RPC to use.

yaml

```
rpcs:
  - name: HASHIO
    network: testnet
    baseUrl: https://testnet.hashio.io/api
    selected: true
  - name: HASHIO
    network: previewnet
    baseUrl: https://previewnet.hashio.io/api
    selected: true
  - name: HASHIO
    network: mainnet
    baseUrl: https://mainnet.hashio.io/api
    selected: true
```

logs

Here, you can specify the path where log files will be stored (path) and the level of logging detail you want (level). For example, setting the level to ERROR will only log error events.

yaml

```
logs:
```

```
path: ./logs
level: ERROR
```

factories

This property lists factory contract IDs (id) and their associated network. Factories are smart contracts that can create other contracts. By listing them here, the application knows which factories it can interact with on each network.

```
yaml
factories:
  - id: 0.0.636690
    network: testnet
```

<details>

<summary>Example configured <code>hsca-config.yaml</code> file ☒ </summary>

```
yaml
defaultNetwork: testnet
networks:
  - name: mainnet
    consensusNodes: []
  - name: previewnet
    consensusNodes: []
  - name: testnet
    consensusNodes: []
accounts:
  - accountId: 0.0.1234
    privateKey:
      key: >-
        302e020100300506032XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      type: ED25519
    network: testnet
    alias: main
    importedTokens: []
mirrors:
  - name: HEDERA
    network: testnet
    baseUrl: https://testnet.mirrornode.hedera.com/api/v1
    selected: true
  - name: HEDERA
    network: previewnet
    baseUrl: https://previewnet.mirrornode.hedera.com/api/v1
    selected: true
  - name: HEDERA
    network: mainnet
    baseUrl: https://mainnet.mirrornode.hedera.com/api/v1
    selected: true
rpcs:
  - name: HASHIO
    network: testnet
    baseUrl: https://testnet.hashio.io/api
    selected: true
  - name: HASHIO
    network: previewnet
    baseUrl: https://previewnet.hashio.io/api
    selected: true
  - name: HASHIO
    network: mainnet
    baseUrl: https://mainnet.hashio.io/api
```

```
    selected: true
logs:
  path: ./logs
  level: ERROR
factories:
  - id: 0.0.636690
    network: testnet
```

</details>

Factory contracts

We provide default contract addresses for factories that we have deployed for anyone to use. They are updated whenever a new version is released.

Contract name	Address	Network
-----	-----	-----
FactoryAddress	0.0.14455068	Testnet
FactoryAddress	0.0.XXXXXX	Previewnet

A factory contract is a specialized type of smart contract designed to generate another smart contract. It serves as a template or blueprint for creating multiple instances of other contracts with similar features but possibly different initial states and parameters.

> For example, the ERC20Factory is a contract that facilitates the deployment of new ERC20 tokens. The factory uses the EIP1167 standard for simply and cheaply cloning contract functionality.

Deploy custom factory contract (optional)

If you want to deploy your own factory contract with custom logic tailored to your specific needs, check out the /contracts folder README for a comprehensive guide.

CLI Flow

<figure><figcaption></figcaption></figure>

Start CLI

Start the CLI tool using the wizard command:

```
bash
accelerator wizard
```

When the CLI is started with the configuration file properly configured, the first action will be to select the account you want to operate with. The list of configured accounts belonging to the default network indicated in the configuration file is displayed by default.

If there are no accounts in the file for the default network, a warning message will be displayed and a list of all the accounts in the file will be displayed.

<figure><figcaption></figcaption></figure>

Main Menu

The main menu will be displayed once you select an account you want to operate with. Choose one of the stablecoin management menu options listed and follow the prompts in your interface. Let's review each option to give you a better understanding of what's available to you:

<details>

<summary>Create a new stablecoin</summary>

This option starts your stablecoin creation process and where you will fill in the details of your stablecoin. Details such as:

1. Set Factory: Before anything else, specify the factory you'll be using. You can find the list of deployed factories in our documentation.
2. Basic Details: Input essential information like the stablecoin's name and symbol.
3. Auto-Renew Account: This is set by default to your current account, enabling the stablecoin creation process.
4. Optional Details: You'll be prompted to add details like decimal count, initial, and max supply. Default values are used if you skip.
5. Token Keys Ownership: Decide whether the smart contract should own all underlying token keys like pause and wipe.
6. KYC: Choose whether to enable KYC. If yes, specify the KYC key and decide whether to grant KYC to the current account during stablecoin creation.
7. Custom Fees: Decide if you want to add custom fees to the token. If yes, specify the fee schedule key.
8. Role Management: For any token keys set to the smart contract, you can grant or revoke roles to other accounts.
9. Proof of Reserve (PoR): Optionally, link your stablecoin to an existing PoR contract or deploy a new one, setting an initial reserve. Note that PoR here is demo-quality.
10. PoR Admin: If deploying a new PoR, your current account will become its admin by default.
11. Proxy Admin Ownership: Choose the account that will own the proxy admin. It can be set to another account or contract.
12. Configuration File: Make sure your configuration file contains the right factory and HederaTokenManager contract addresses.
13. Execute: Finally, the CLI will use the configuration file to submit the request, and your stablecoin will be created.
14. Additional Contracts: Users who wish to use their own contracts can update the configuration file with the new factory contract ID.

Refer to the ChainLink documentation for more on PoR feeds.

</details>

<details>

<summary>Manage imported tokens</summary>

Stablecoins that we have not created with our account but for which we have been assigned one or several roles must be imported in order to operate them.

1. Add token
2. Refresh token
3. Remove token

</details>

<details>

<summary>Operate with an existing stablecoin</summary>

Once a stablecoin is created or added, you can operate with it. The following list contains all the possible operations a user can perform if they have the appropriate role.

1. Send Tokens: Transfer tokens to other accounts.
2. Cash In: Mint tokens and transfer them to an account. If you have linked a PoR Feed to your stablecoin, this operation will fail in two cases:
 1. If you try to mint more tokens than the total Reserve (1 to 1 match between the token's total supply and the Reserve)
 2. If you try to mint tokens using more decimals than the Reserve has, for instance, minting 1.001 tokens when the Reserve only has two decimals.

> 🗨 This DOES NOT mean that a stablecoin can not have more decimals than the Reserve, transfers between accounts can use as many decimals as required.

<!-->

3. Check Details: Retrieve the stablecoin's details.
4. Check Balance: Retrieve an account's balance.
5. Burn Tokens: Burn tokens from the treasury account.
6. Wipe Tokens: Burn tokens from any account.
7. Rescue Tokens: Use the smart contract to transfer tokens from the treasury to a rescue account.
8. Rescue HBAR: Similarly, transfer HBAR from the treasury to a rescue account via smart contract.
9. Freeze Management: Freeze or unfreeze an account and verify its status. If an account is frozen, it will not be able to transfer any tokens.
10. Role Management: Administrators of a stablecoin can manage user roles from this menu. They can grant, revoke, edit (manage the supplier allowance), and check roles.
 1. The available roles are:
 1. CASHINROLE
 2. BURNROLE
 3. WIPEROLE
 4. RESCUEROLE
 5. PAUSEROLE
 6. FREEZEROLE
 7. KYCROLE
 8. DELETEROLE

<!-->

11. Configuration: The last option in the menu allows for dual configuration management of both the stable coin and its underlying token. For the stablecoin, you can upgrade its contract and change its proxy admin. For the token, admins can edit attributes like name, symbol, and keys. To transfer proxy admin ownership, the current owner invites a new account ID and can cancel before acceptance. Once accepted, the ownership change is finalized.
12. <mark style="color:red;">Danger Zone</mark>: This option groups high-risk stablecoin operations that either impact all token owners or are irreversible. These are placed in a sub-menu to prevent accidental execution.

1. Pause/Unpause: Suspend or resume all token operations.
 2. Delete Token: Permanently remove a token. This action is irreversible.

> 🗨 Take caution when using operations in the "Danger Zone" as they have significant impacts and may not be reversible.

See the repo for more details.

</details>

<details>

<summary>List Stablecoins</summary>

This option displays all the stable coins the user has created or added.

</details>

<details>

<summary>Configuration</summary>

This last option allows the user to display the current configuration file, modify the configuration path, change the default network and manage:

Accounts: Allows the user to change the current account, see all configured accounts and also add new accounts and remove existing ones.

Mirror nodes: Allows the user to change the current mirror node, see all configured mirror nodes for the selected Hedera network, add new mirror nodes and remove existing ones except for the one that is being used.

JSON-RPC-Relay services: Allows the user to change the current JSON-RPC-Relay service, see all configured services for the selected Hedera network, add new JSON-RPC-Relay services and remove existing ones except for the one that is being used.

Factory: Allows the user to change the factory id of the selected Hedera network in the configuration file, to upgrade the factory's proxy, to change the factory's proxy admin owner account and, finally, to view the current factory implementation contract address as well as the factory owner account previously commented.

</details>

Create Stablecoin

Start the CLI tool using the wizard command:

```
bash
accelerator wizard
```

Select "Create a new stablecoin" and proceed through the prompts to fill in the details of your new stablecoin.

<figure><figcaption><p>CLI main menu</p></figcaption></figure>

<figure><figcaption><p>CLI stablecoin creation prompts</p></figcaption></figure>

```
{% embed url="https://youtu.be/gh7fVX1iY0?si=nwVuA2YkV-cJje2A" %}
How to Issue Stablecoins on Hedera: Create a Stablecoin\
by Developer Advocate: Michiel Mulders
{% embed %}
```

Operate Stablecoin

Start the CLI using the wizard command:

```
bash
accelerator wizard
```

Select the "Operate with an existing stablecoin" option and proceed through the prompts to operate your stablecoin.

<figure><figcaption><p>CLI stablecoin operation menu</p></figcaption></figure>

{% embed url="https://youtu.be/41ag-y9cYck?si=R0-6P9TQJLXJdgyQ" %}
How to Issue Stablecoins on Hedera: Stablecoin Administration\
by Developer Advocate: Pathorn Tengkiattrakul
{% endembed %}

▫ Here is a list of stablecoin operations.

Configure Proof of Reserve (PoR)

Start the CLI using the wizard command:

```
bash
accelerator wizard
```

Select the "Create a new stablecoin" option, then proceed through the prompts.

<figure><figcaption></figcaption></figure>

{% embed url="https://youtu.be/a7sNXD5GKWA?si=iuPkensCQu23P0" %}
How to Issue Stablecoins on Hedera: Create a Stablecoin\
by Developer Advocate: Greg Scullard
{% endembed %}

Additional Resources

{% embed url="https://portal.hedera.com" %}

{% embed url="https://docs.hedera.com/hedera/support-and-community/glossary" %}
Hedera and Web3 Glossary
{% endembed %}

{% embed url="https://www.hashpack.app/download" %}
Hedera Non-Custodial Wallet
{% endembed %}

{% embed url="https://hashscan.io" %}
Hedera Network Explorer
{% endembed %}

{% embed url="https://www.npmjs.com/package/@hashgraph/stablecoin-npm-cli" %}

core-concepts.md:

Core Concepts

Introduction

As the adoption of cryptocurrencies grows, a common obstacle deterring new adoption by users is price volatility, particularly when considering these assets for everyday transactions. Stablecoins offer a solution to this volatility by maintaining a consistent value, pegged to traditional assets like the US Dollar. Prominent examples include Tether (USDT), USD Coin (USDC), and Binance USD (BUSD). The stablecoin landscape also features asset-backed options such as MakerDAO's DAI, which is crypto-secured, and Paxos Gold (PAXG), backed by physical gold. These stable assets have not only transformed crypto trading by offering a secure place to store value temporarily but also have the potential to revolutionize e-commerce by providing a stable medium for online transactions.

> A stablecoin is a specialized form of cryptocurrency engineered to maintain a stable value by pegging it to an external asset, such as a fiat currency like the US Dollar or a commodity like gold. By doing so, stablecoins seek to combine the programmability and ease of transfer inherent in cryptocurrencies with the price stability typically associated with traditional currencies. This low-volatility characteristic makes stablecoins particularly useful for transactions, cross-border payments, and as a stable asset within decentralized finance ecosystems.

Despite its many advantages, Hedera currently lacks a ready-to-use stablecoin framework. Stablecoin Studio aims to fill that void by offering a comprehensive stablecoin solution tailored to Hedera's architecture. Developers will be equipped with a suite of resources, including essential tools, documentation, and sample code, allowing them to create applications that make use of stablecoins, such as digital wallets. The ultimate goal is to facilitate the seamless integration of stablecoins into a variety of platforms and applications, thereby boosting Hedera's utility and adoption.

Stablecoin Studio

Stablecoin Studio is an open-source solution that simplifies and enhances the granularity of access control and permission management when issuing stablecoins using Hedera network services. Utilizing a hybrid solution, the platform leverages both Hedera Token Service (HTS) and Hedera Smart Contract Service (HSCS), offering interoperability with Solidity Smart Contracts. This adds an extra layer of flexibility and capability for stablecoin issuers. As an all-in-one toolkit, this project enables stablecoin issuers to easily deploy applications and oversee operations via a comprehensive management toolkit, streamlining digital asset operations.

The toolkit offers proof-of-reserve (PoR) functionality that utilizes existing systems or on-chain oracles to bolster the ability to provide transparency in disclosure while seamless custody provider integrations ease development and reduce time-to-market.

Complemented by advanced Hedera-native Know Your Customer (KYC) / Anti-Money Laundering (AML) account flags and integrated service provider hooks, Stablecoin Studio gives issuers new ways to manage compliance and security.

Core Components

The core components of Stablecoin Studio include these four components that help simplify stablecoin creation and management:

|--|--|--|

	data-hidden data-target data-type="content-ref"></th><th> <th data-kind="ghost"></th>	
card-cover data-type="files"></th></tr></thead><tbody><tr><td>TypeScript SDK</td><td>The SDK implements the features for creating, managing, and operating stablecoins. The SDK interacts with the Smart Contract and exposes an API to be used by client-facing applications.</td><td>https://github.com/hashgraph/stablecoin-studio/blob/main/sdk/README.md</td><td>stablecoin-typescript-icon.png</td></tr><tr><td>Web UI Application</td><td>A visually rich platform that adds a layer of user-friendly interaction for creating, managing, and operating tokens. A dApp developed in React. Uses the SDK-exposed API.</td><td>web-ui-application.md</td><td>stablecoin-web-ui-Icon.png</td></tr><tr><td>Command Line Interface (CLI)</td><td>A command line interface (CLI) tool for creating, managing, and operating stablecoins. Uses the SDK-exposed API.</td><td>cli-management.md</td><td>stablecoin-command-line-tool.png</td></tr><tr><td>Smart Contracts</td><td>A set of well-audited, open-source, smart contracts ready to be deployed for creating and managing stablecoins on Hedera network.</td><td>https://github.com/hashgraph/stablecoin-studio/blob/main/contracts/README.md</td><td>smart-contracts-icon.png</td></tr></tbody></table>		

Robust Administration and Compliance Configurations

Stablecoin Studio includes Hedera-native token administration functionalities, enabling issuers to easily burn, mint (cash-in), freeze, wipe, and pause stablecoins. Account-based permissions, like native KYC account flags, provide compliance configurations when connecting Stablecoin Studio to custody providers and KYC/AML services.

Hedera Native Token Administration Functionalities

	Functionality	Description
<h4>CashIn (Mint)</h4>		Issuers can effortlessly create new tokens, increasing the total supply of the stablecoin. Minting is often subject to regulatory compliance and internal governance.
<h4>Burn</h4>		The platform allows issuers to reduce the overall token supply by 'burning' or destroying tokens, usually in a controlled and auditable manner.
<h4>Freeze/Unfreeze</h4>		For various compliance-related or operational reasons, Stablecoin Studio provides the option to freeze tokens, making them non-transferable until further actions are taken.
<h4>Pause/Unpause</h4>		The platform includes a <code>pause</code> functionality, enabling issuers to temporarily halt all token-related activities in emergencies or for scheduled maintenance.
<h4>Wipe</h4>		Stablecoin Studio allows issuers to <code>wipe</code> or delete tokens from specific accounts, mainly for regulatory or security reasons.

Account-Based Permissions and Compliance Features

Compliance Mechanisms	Description
Native KYC Account Flags	Incorporate KYC checks directly into the token issuance and management process. These flags serve as compliance markers, indicating whether an account has been verified for KYC requirements.
Integration with Custody Providers	Stablecoin Studio is designed to connect seamlessly with third-party custody solutions, offering an extra layer of security for asset management.
KYC/AML Service Connections	Through native account flags and API integrations, Stablecoin Studio can be easily hooked into existing KYC/AML solutions, streamlining the compliance workflow.

Ecosystem Integrations

The Stablecoin Studio SDK provides integrations into third-party providers who can fully support your stablecoin offering on the public Hedera network. These providers offer solutions for KYC/AML, custody, wallets, infrastructure management, smart contract monitoring, regulatory compliance, and more.

Custody Providers

<p>Integration in progress</p> <p>Zodia specializes in institutional-grade digital asset custody, offering cutting-edge tech within a compliant framework. They primarily cater to corporate investors and institutions.</p> <p>LEARN MORE</p> <p>stablecoin-zodia-logo.png</p>	<p>Integration in progress</p> <p>DFNS offers a wallet-as-a-service infrastructure targeting crypto developers, allowing developers to focus on their applications by handling private key management.</p> <p>LEARN MORE</p> <p>stablecoin-dfns-logo.png</p>
<p>Integration in progress</p> <p>Hex Trust offers bank-grade digital asset custody. Led by experienced finance and tech experts, their Hex Safe™ platform provides an array of custody solutions.</p> <p>LEARN MORE</p> <p>hex-trust2.png</p>	

KYC/AML Service Providers

<p>Coming soon</p> <p>Elliptic offers a suite of compliance solutions designed to assist financial institutions. Services include: transaction monitoring for KYC/AML purposes, identity verification, and risk assessment.</p> <p>LEARN MORE</p> <p>stablecoin-elliptic-</p>	

logo.png</td><td>https://www.elliptic.co/</td></tr><tr><td align="center"><p>Coming soon</p><p></p></td><td>Merkle Science provides transaction monitoring and intelligence solutions for cryptoasset service providers, financial institutions, and government agencies to detect, investigate, and prevent money laundering.</td><td align="center"><p></p><p>LEARN MORE</p></td><td>stablecoin-merkle-science-logo.png</td><td>https://www.merklescience.com/</td></tr></tbody></table>

Oracle Provider

<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th></th><th></th><th align="center"></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Available</td><td></td><td><td>HashPort Pro offers an advanced middleware toolkit for enterprises deploying applications on Hedera, using its Validator Swarm for transaction verification. Axiom, a feature of HashPort Pro, is an on-chain oracle linking to banking and fintech for price and reserve data.</td><td align="center"><p></p><p>CONTACT FOR INTEGRATION</p></td><td>stablecoin-pro-axiom-logo.png</td><td>https://www.hashport.network/contact/</td></tr></tbody></table>

Proof-of-Reserve (PoR) for Treasury Management

To ensure the security and transparency of your stablecoin, Stablecoin Studio offers a proof-of-reserve (PoR) feature. This feature serves two key objectives: preventing under-collateralization and providing transparent asset backing for your stablecoin. The PoR system employs automated verification, where off-chain assets are continually monitored through oracles or similar mechanisms. These monitoring systems subsequently update a dedicated reserve contract to reflect changes in the asset value accurately.

Before you mint new tokens, a smart contract automatically checks this reserve contract. Minting is authorized only when sufficient reserve assets are available to back the new tokens. For instance, let's say you have enabled PoR with a reserve of 1,000 units. You can comfortably mint up to 900 tokens. However, if you attempt to mint an additional 100 tokens, the smart contract will halt the process due to insufficient reserves. You also have the flexibility to update these reserves either manually or programmatically as per your operational requirements.

Lastly, when using Stablecoin Studio, you can link an existing Oracle-fed reserve contract or let the platform automatically deploy a new one. Both options aim to guarantee the stability of your stablecoin and foster user trust.

<figure><figcaption><p>Proof of Reserve: How does it work?</p></figcaption></figure>

##

Stablecoin Studio: How It All Works Together

At a functional level, Stablecoin Studio provides a comprehensive architecture that enables developers to create, manage, and operate stablecoins on the Hedera network using pre-built “factory smart contracts” audited by CertiK. The architecture consists of several components that work together to facilitate the deployment and management of stablecoins:

1. **SDK:** The Software Development Kit (SDK) provides a set of tools and libraries for developers to build and deploy stablecoin applications on the Hedera network. It simplifies the process of interacting with Stablecoin Studio’s “factory contracts” used to create and manage stablecoins.
2. **CLI:** The Command Line Interface (CLI) offers a user-friendly way for developers to interact with the SDK and manage stablecoins. It allows developers to perform various operations, such as creating, managing, and operating stablecoins using simple commands.
3. **Web UI:** The User Interface (UI) component provides a graphical interface for developers and users to interact with stablecoins on the Hedera network. It simplifies the process of managing and operating stablecoins for non-technical users.

When you initiate the creation process using the SDK, CLI, or Web UI, two smart contracts are deployed: The Hedera Token Manager Proxy and the Hedera Token Manager. The former oversees the ownership of the latter, which in turn is responsible for managing the permissions and functionalities of your stablecoin.

As a stablecoin admin/delegate, you’ll manage your stablecoin using the Hedera Token Manager smart contract, offering granular control and flexibility. End-users will not need to navigate the complexities of smart contracts – they can interact directly with your stablecoin using the Hedera Token Service (HTS), ensuring a more convenient, cost-effective, and scalable experience.

<figure><figcaption><p>Stablecoin Studio Work Flow/Process</p></figcaption></figure>

README.md:

Stablecoin Studio

web-ui-application.md:

Web UI Application

The Stablecoin Studio Web User Interface (Web UI) is designed to provide a more intuitive and user-friendly experience for managing stablecoins. Developed using React, this interface offers a visual representation of all our Web UI capabilities, from initial setup to advanced administration. Whether you're a newcomer looking for an easier entry point or a seasoned developer seeking a more efficient way to manage stablecoins, this documentation offers the insights and resources to harness the full power of our React-based Web UI.

Table of Contents

1. Interactive Demo
2. Prerequisites
3. Installation
4. Start Web UI

5. Create Stablecoins
6. Operate Stablecoins
7. Manage Roles
8. Stablecoin Details
9. Additional Resources

Interactive Demo

Get firsthand experience with Stablecoin Studio's capabilities using the open-source, React-based demo application sandbox. The sandbox application is built using Stablecoin Studio's TypeScript SDK.

```
<figure><figcaption></figcaption></figure>
```

After setting up a Hedera testnet account, explore creating and managing stablecoins on Hedera through the interactive demo and follow along. Let's get started and explore the three paths for launching the Stablecoin Studio web application:

```
<table data-view="cards"><thead><tr><th></th><th></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td><ol><li><a href="https://stablecoinstudio.com/"><strong>Code Sandbox</strong></a></li></ol></td><td>For a quick and effortless start, you can use the pre-configured Code Sandbox environment directly from StablecoinStudio.com. This approach requires no setup and provides a fully functional demo application.</td><td><a href="../../.gitbook/assets/stablecoin-code-sandbox-icon.png">stablecoin-code-sandbox-icon.png</a></td><td><a href="https://www.stablecoinstudio.com">https://www.stablecoinstudio.com</a></td></tr><tr><td><ol start="2"><li><a href="https://gitpod.io/new/#https://github.com/hashgraph/stablecoin-studio"><strong>GitPod Instance</strong></a></li></ol></td><td>Another easy way to get started is by launching a GitPod instance, which automates the initial setup and lets you dive into the application immediately. You can skip the <em>prerequisites</em> and <em>installation</em> steps.</td><td><a href="../../.gitbook/assets/stablecoin-gitpod-icon.png">stablecoin-gitpod-icon.png</a></td><td><a href="https://gitpod.io/new/#https://github.com/hashgraph/stablecoin-studio">https://gitpod.io/new/#https://github.com/hashgraph/stablecoin-studio</a></td></tr><tr><td><ol start="3"><li><a href="web-ui-application.md#prerequisites"><strong>Local Environment</strong></a></li></ol></td><td>A more technical method to build and install on your local machine. This guide focuses on this method, walking you through the steps needed to set up your local environment. Start from the first step: Prerequisites.</td><td><a href="../../.gitbook/assets/stablecoin-local-environment-Icon.png">stablecoin-local-environment-Icon.png</a></td><td><a href="web-ui-application.md#prerequisites">#prerequisites</a></td></tr></tbody></table>
```

Prerequisites

NodeJS >= 18.13
 Solidity >= 0.8.16
 TypeScript >= 4.7
 Git Command Line
 Hedera Testnet Account
 HashPack Wallet

Install Stablecoin Studio

Open a new terminal and navigate to your preferred directory location where you want your Stablecoin Studio project to live. Clone the repo, cd in to the cloned directory, and install dependencies:

```
{% code fullWidth="false" %}
bash
git clone https://github.com/hashgraph/stablecoin-studio.git
cd stablecoin-studio
npm install
```

```
{% endcode %}
```

cd in to the web directory:

```
bash
cd web
```

Environment variables

Before we can start the web application, we need to configure the environment variables. From the root of the web project workspace, rename the .env.sample file to .env and configure the variables required to run the application. For the purposes of this demo, we will need:

1. The factory contract ID.
2. The REST API testnet endpoint.
3. The JSON-RPC relay testnet endpoint.

The .env file contains the following parameters:

Environment Variable	Description
REACTAPPLLOG LEVEL	Log level for the application. The default value is <code>ERROR</code> . Acceptable values are <code>ERROR</code> , <code>WARN</code> , <code>INFO</code> , <code>HTTP</code> , <code>VERBOSE</code> , <code>DEBUG</code> , and <code>SILLY</code> in order of priority (highest to lowest).
REACTAPPFATORIES	JSON array with a factory contract ID in Hedera format <code>0.0.XXXXX</code> per environment.
REACTAPPMIRRORNODE	The var must be a unique mirror node service for Hedera network, and this is the service that would be used when the UI starts. If the service doesn't require an API key to authorize requests the <code>APIKEY</code> and <code>HEADER</code> properties must remain empty. Here is a list of mirror node endpoints.
REACTAPPRPCNODE	The var must be a unique RPC node service for Hedera network, and this is the service that would be used when the UI starts. If the service doesn't require an API key to authorize requests the <code>APIKEY</code> and <code>HEADER</code> properties must remain empty. Here is a list of RPC providers.
GENERATESOURCEMAP	This is a proprietary Create React App configuration. You can read more information in its Create React App documentation .

<details>

<summary>Example configured `.env` file ☒ </summary>

```
{% code title=".env" overflow="wrap" fullWidth="false" %}
```

```
bash
REACTAPPPLOGLEVEL=ERROR
REACTAPPPFACTORIES='[{"Environment":"testnet","STABLECOINFACTORYADDRESS":"0.0.467
235"}]'
REACTAPPMIRRORNODE='[{"Environment":"testnet","BASEURL":"https://
testnet.mirrornode.hedera.com/api/v1/", "APIKEY": "", "HEADER": ""}]'
REACTAPPPRPCNODE='[{"Environment":"testnet","BASEURL":"https://
testnet.hashio.io/api", "APIKEY": "", "HEADER": ""}]'
GENERATESOURCEMAP=false

{% endcode %}

</details>
```

Start the Web UI

Once you have configured your environment variables, start the web UI from the web directory:

```
bash
npm run start
```

If the application is successfully run, the web application interface will open in a new browser:

```
<figure><figcaption><p>http://localhost:3000/</p></figcaption></figure>
```

Click "Connect your wallet" and select the wallet (HashPack or MetaMask) and network you want to interact with. For the purposes of this demo, we will use HashPack and select Testnet.

```
<div>
```

```
<figure><figcaption></figcaption></figure>
```

```
<figure><figcaption></figcaption></figure>
```

```
</div>
```

Now that your project is set up and the web application is running let's create our first stablecoin!

Create Stablecoins

Basic details (Required)

To initiate the creation of your stablecoin, head to the top of the interface and click on the `<mark style="background-color:purple;">+</mark>` symbol. From the options that appear, select "Create coin." The required fields for basic details will be displayed:

HederaTokenManager impl.: By default, this is set to a factory contract ID provided by Hedera, in the format 0.0.XXXXXX. Advanced users have the option to deploy their own factory contract implementation.

Name: This is where you name your new stablecoin, for example, "NewStableCoin."
Symbol: Enter a symbol to represent your stablecoin, like "\$NSC."

<figure><figcaption></figcaption></figure>

Optional details

While the basic details are mandatory, you also have an option to further specify:

Initial Supply: You can expand on the initial number of tokens that will be minted.

Max Supply: If you chose 'Finite' in the 'Supply Type,' you might want to set an upper limit.

Decimals: You can set additional decimal places if you need more precision.

<figure><figcaption></figcaption></figure>

Manage permissions

When creating a stablecoin, you have multiple key configuration options. One of those is the "Underlying Token's Keys Definition." This determines which accounts control various operations of the stablecoin, such as who can approve KYC checks or which account can change the coin supply. You have the flexibility to set these keys to be controlled by the stablecoin's initial smart contract, assign them to a different key, or even leave them undefined.

If the KYC key is tied to the smart contract and the supply key isn't tied to the account that creates the stablecoin, you can opt to automatically grant KYC verification to the account creating the stablecoin at the time of its creation.

As for ownership settings, by default, the account that initiates the stablecoin creation also becomes the stablecoin proxy admin owner. However, you're not locked into this default setup. You can alter this by specifying a different account ID during creation. This could be any account, including specialized accounts like a timelock controller for scheduled operations or a cold wallet for enhanced security.

<figure><figcaption></figcaption></figure>

Proof-of-reserve (PoR)

Choose if the stablecoin will be linked to a Proof of Reserve (PoR) contract. You can either use an existing PoR contract address or create a new one using the demo implementation included in the project and setting an initial reserve amount.

{% hint style="warning" %}

Warning: You can change the PoR contract address at any time, but exercise caution. Altering the address can affect your stablecoin's cash-in functionality, as it refers to a new reserve verification contract. In cases where the new contract has a lower reserve than the previous one, minting new tokens may become restricted.

{% endhint %}

For those using the project's demo PoR contract, you also have the option to modify its reserve amount. This can be done via the PoR admin account used during the stablecoin's deployment. Because the reserve can be changed arbitrarily in the demo, it's intended for demonstration purposes only.

<figure><figcaption></figcaption></figure>

Review

Final validation before creating the stablecoin. Review the stablecoin details and click the "Create stablecoin" button. Validate "Execute Smart Contract" and "Associate Token" transactions in your wallet. Once the stablecoin is created, it will be added to the drop-down list of coins you can access (with the account you used to create the stablecoin).

<figure><figcaption></figcaption></figure>

Operate Stablecoins

To operate your stablecoin, connect your wallet to the platform. After successful authentication, select the stablecoin you wish to interact with from the drop-down list of available coins. Once the stablecoin information is loaded, navigate to the "Operations" tab.

You'll see a variety of actions and your accessible operations will be tied to the roles assigned to your account for the chosen stablecoin. Here's a quick rundown of what each operation allows:

- Cash In: Deposit assets into your stablecoin account.
- Burn: Permanently remove specific tokens from circulation.
- Get Balance: View the current balance of your stablecoin account.
- Rescue: Recover tokens in unique scenarios.
- Rescue HBAR: Specialized recovery for HBAR.
- Wipe: Clear particular stablecoin balances.
- Freeze Account: Temporarily disable transaction capabilities for an account.
- Unfreeze: Lift the freeze status from an account.
- Check Account Frozen Status: Verify whether an account is frozen.
- Grant KYC: Approve an account for KYC verification.
- Revoke KYC: Remove previously granted KYC approval.
- Check KYC: Confirm the KYC status of an account.
- Danger Zone: Access to operations that carry higher risk, generally because they affect every token owner (PAUSE) or can not be rolled back (DELETE).

To carry out an operation, simply click on the corresponding button and follow the on-screen prompts. The platform will automatically perform the operation based on the capabilities your account has been assigned.

The "Operations" tab in Stablecoin Studio is your hub for managing every aspect of your stablecoin, so make sure you're familiar with the roles and capabilities assigned to your account to leverage the suite of operations available to you fully.

<figure><figcaption></figcaption></figure>

Manage Roles

In Stablecoin Studio, role management is a pivotal feature that gives you control over various functions related to your stablecoin. If your account has been designated with the "Admin Role," you unlock the capability to manage other roles for your stablecoin, making governance easier and more secure.

Roles you can manage include:

Cash In: Permits an account to deposit or 'cash in' assets.
Burn: Authorizes an account to remove tokens from circulation permanently.
Wipe: Allows an account to clear specific balances.
Rescue: Grants the ability to recover tokens in special circumstances.
Pause: Enables stopping all transactions temporarily, which is useful in emergency situations.
Freeze: Authorizes freezing specific accounts, disabling their ability to transact.
Delete: Allows the removal of accounts or certain data, irreversible.
Admin Role: Provides overarching administrative privileges, often reserved for key governance participants.

Connect your wallet and select the stablecoin from the drop-down list you want to interact with. Once the stablecoin information loads, head to the "Role management" tab.

<figure><figcaption></figcaption></figure>

Stablecoin Details

This menu option displays stablecoin details and allows the user to update some token properties, like the name, symbol, and keys. Clicking on the pencil icon located at the top right side of the screen turns the information page into a form where these properties can be modified by the user.

🎉 Congrats on creating your first stablecoin with Stablecoin Studio! View the transaction details on HashScan by looking up your new token ID or clicking the HashScan Explorer link from the "Token ID" field.

<div>

<figure><figcaption><p>HashScan Explorer link</p></figcaption></figure>

<figure><figcaption><p>HashScan</p></figcaption></figure>

</div>

Additional Resources

{% embed url="https://portal.hedera.com/" %}
Hedera Developer Portal (Testnet Accounts)
{% embed %}

{% embed url="https://docs.hedera.com/hedera/support-and-community/glossary" %}
Hedera and Web3 Glossary
{% embed %}

{% embed url="https://www.hashpack.app/download" %}
Hedera Non-Custodial Wallet
{% embed %}

{% embed url="https://hashscan.io" %}
Hedera Network Explorer
{% embed %}

```
{% embed url="https://www.npmjs.com/package/@hashgraph/stablecoin-npm-sdk" %}
```

hedera-consensus-service-api.md:

```
---
description: >-
  The Hedera Consensus Service (HCS) gRPC API is a public mirror node managed by
  Hedera. It offers the ability to subscribe to HCS topics and receive messages
  for the topic subscribed.
---
```

Hedera Consensus Service gRPC API

```
{% hint style="warning" %}
Important Notice: Deprecation of the Insecure Hedera Consensus Service (HCS)
Mirror Node Endpoints
```

We are phasing out the legacy Hedera Consensus Service (HCS) mirror node endpoints. The APIs have transitioned from the legacy hcs.

<env>.mirrornode.hedera.com:5600 endpoints to the new
<env>.mirrornode.hedera.com:443 endpoints.

For more details, please read our blog post announcement.
{% endhint %}

```
{% hint style="info" %}
HCS Mirror Node Endpoints:\
PREVIEWNET: previewnet.mirrornode.hedera.com:443\
TESTNET: testnet.mirrornode.hedera.com:443\
MAINNET: mainnet.mirrornode.hedera.com:443
{% endhint %}
```

```
{% hint style="warning" %}
Requests for the public mainnet mirror node are throttled at 100 requests per
second (rps). This may change in the future depending upon performance or
security considerations. At this time, no authentication is required.
{% endhint %}
```

Community-supported mirror node information can be found here:

```
{% content-ref url="../networks/community-mirror-nodes.md" %}
community-mirror-nodes.md
{% endcontent-ref %}
```

Build a Mirror Node Client

If you building your client with a predefined Hedera network (previewnet, testnet, mainnet), you do not need to define the mirror client as it is built in. If you would like to modify the mirror node client, you can use <mark style="color:purple;">Client.<network>.setMirrorNetwork(<network>)</mark>.

```
{% tabs %}
{% tab title="Java" %}
{% code title="Java" %}
java
// You will need to upgrade to v2.0.6 or higher
Client client = Client.forMainnet();
client.setMirrorNetwork(Collections.singletonList("mainnet.mirrornode.hedera.com:443"))
{% endcode %}
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
// You will need to upgrade to v2.0.23 or higher
const client = Client.forMainnet()
client.setMirrorNetwork("mainnet.mirrornode.hedera.com:443")

{% endtab %}

{% tab title="Go" %}
go
hedera.ClientForMainnet()
client.SetMirrorNetwork([]string{"mainnet.mirrornode.hedera.com:443"})

{% endtab %}
{% endtabs %}

{% hint style="warning" %}
Concurrent Subscription Limit\
A single client can make a maximum of 5 concurrent subscription calls per
connection.
{% endhint %}
```

Subscribe to a topic

Please click the link below to see how you can subscribe to a topic.

```
{% content-ref url="sdks/consensus-service/get-topic-message.md" %}
get-topic-message.md
{% endcontent-ref %}
```

README.md:

```
---
cover: ../.gitbook/assets/Hero-Desktop-Tooling2022-12-07-021130ayix (1) (1).webp
coverY: -120
---
```

SDKs & APIs

rest-api.md:

```
---
description: The mirror node REST API offers the ability to query transaction
information.
---
```

REST API

Hedera Mirror Nodes store the history of transactions that occurred on mainnet, testnet, and previewnet. Each transaction generates a record that is stored in a record file. The transaction contents can be accessed by the mirror node REST APIs

To make a request, use the network endpoint and the REST API of choice. For example, to get a list of transactions on mainnet you would make the following request.

```
{% embed url="https://mainnet.mirrornode.hedera.com/api/v1/transactions" %}

<details>
```

<summary>Hedera Mirror Node Swagger UI environments</summary>

\

Mainnet

Testnet

Previewnet

</details>

{% hint style="info" %}

MAINNET BASEURL\

<https://mainnet.mirrornode.hedera.com/>

TESTNET BASEURL\

<https://testnet.mirrornode.hedera.com/>

PREVIEWNET BASEURL\

<https://previewnet.mirrornode.hedera.com/>

You may also check out Validation Cloud, DragonGlass, Arkhia or Ledger Works as alternatives.

{% endhint %}

{% hint style="warning" %}

Public mainnet mirror node requests per second (RPS) are currently throttled at 50 per IP address. These configurations may change in the future depending on performance or security considerations. At this time, no authentication is required.

{% endhint %}

Accounts

The accounts object represents the information associated with an account and returns a list of account information.

Account IDs take the following format: 0.0.\<account number>.

Example: 0.0.1000

Account IDs can also take the account number as an input value. For example, for account ID 0.0.1000, the number 1000 can be specified in the request.

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/accounts" method="get" %}

<https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml>

{% endswagger %}

Response Details

Response Item	Description

account	The ID of the account
allowances	The allowances granted to this account
alias	RFC4648 no-padding base32 encoded

account alias		
auto\renew\period		The period in which the account will auto
renew		
balance		The timestamp and account balance of the
account		
created\timestamp		The timestamp for the creation of that
account		
decline\reward		Whether or not the account has opted to
decline a staking reward		
deleted		Whether the account was deleted or not
ethereum\nonce		The ethereum transaction nonce associated
with this account		
evm\address		A network entity encoded as an EVM
encoded hex		
expiry\timestamp		The expiry date for the entity as set by
a create or update transaction		
key		The public key associated with the
account		
links.next		Hyperlink to the next page of results
max\automatic\token\associations		The number of automatic token associations,
if any		
memo		The account memo, if any
nfts		List of nfts informations belonging to
this account		
pending\reward		The account's pending staking reward that
has not been transferred to the account		
receiver\sig\required		Whether or not the account requires a
signature to receive a transfer into the account		
rewards		List of rewards which of the account
staked\account\id		The account ID the account is staked to,
if set		
staked\node\id		The node ID the account is staked to, if
set		
stake\period\start		The start of the staking period
tokens		The tokens and their balances associated
to the specified account		

Optional Filtering [](#optional-filtering)

Operator	Example
Description	
lt (less than)	/api/v1/accounts?account.id=lt:0.0.1000
Returns account IDs less than 1000	
lte (less than or equal to)	/api/v1/accounts?account.id=lte:0.0.1000
Returns account IDs less than or equal to 1000	
gt (greater than)	/api/v1/accounts?account.id=gt:0.0.1000
Returns account IDs greater than 1000	
gte (greater than or equal to)	/api/v1/accounts?account.id=gte:0.0.1000

| Returns account IDs greater than or equal to 1000
 |
 | order (order asc or desc values) |
 <p><code>/api/v1/accounts?order=asc</code></p><p></p><p><code>/api/v1/accounts?order=desc</code></p> | <p>Returns account information in ascending order</p><p>Returns account information in descending order</p> |

Additional Examples

| Example Requests
 | Description
 |

/api/v1/accounts?account.id=0.0.1001
Returns the account information of account 1001
/api/v1/accounts?account.balance=gt:1000
Returns all account information that have a balance greater than 1000 tinybars
/api/v1/accounts?account.publickey=2b60955bcbf0cf5e9ea880b52e5b63f664b08edf6ed15e301049517438d61864
2b60955bcbf0cf5e9ea880b52e5b63f664b08edf6ed15e301049517438d61864 public key
/api/v1/accounts/2?transactionType=cryptotransfer
Returns the crypto transfer transactions for account 2.

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}/allowances/crypto" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}/tokens" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}/nfts" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}/allowances/tokens" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
```

```
mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/accounts/{idOrAliasOrEvmAddress}/rewards" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Balances

The balance object represents the balance of accounts on the Hedera network. You can retrieve this to view the most recent balance of all the accounts on the network at that given time. The balances object returns the account ID and the balance in hbars. Balances are checked on a periodic basis and thus return the most recent snapshot of time captured prior to the request.

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/balances" method="get"
%}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

timestamp	The seconds.nanoseconds of the timestamp at which the list of balances for each account are returned
balances	List of balances for each account
account	The ID of the account
balance	The balance of the account
tokens	The tokens that are associated to this account
tokens.token\id	The ID of the token associated to this account
tokens.balance	The token balance for the specified token associated to this account
links.next	Hyperlink to the next page of results

Optional Filtering

Operator	Example
Description	

lt (less than)	/api/v1/balances?account.id=lt:0.0.1000
Returns the balances of account IDs less than 1,000	
lte (less than or equal to)	/api/v1/balances?

account.id=lte:0.0.1000
 | Returns the balances account IDs less than or equal to 1,000
 |
 | gt (greater than) | /api/v1/balances?account.id=gt:0.0.1000
 | Returns the balances of account IDs greater than to 1,000
 |
 | gte (greater than or equal to) | /api/v1/balances?
 account.id=gte:0.0.1000
 | Returns the balances of account IDs greater than or equal to 1,000
 |
 | order (order asc or desc values) |
 <p><code>/api/v1/balances?order=asc</code></p><p></p><p><code>/api/v1/balances?
 order=desc</code></p> | <p>Lists balances in ascending order</p><p></p><p>Lists
 balances in descending order</p> |

Additional Examples

| Example Requests
 | Description
 |

/api/v1/balances?account.id=0.0.1000
Returns balance for account ID 1,000
/api/v1/balances?account.balance=gt:1000
Returns all account IDs that have a balance greater than 1000 tinybars
/api/v1/balances?timestamp=1566562500.040961001
Returns all account balances referencing the latest snapshot that occurred
prior to 1566562500 seconds and 040961001 nanoseconds
/api/v1/balances?account.publickey=2b60955bcbf0cf5e9ea880b52e5b6
3f664b08edf6ed15e301049517438d61864
2b60955bcbf0cf5e9ea880b52e5b63f664b08edf6ed 15e301049517438d61864 public key

Transactions

The transaction object represents the transactions processed on the Hedera network. You can retrieve this to view the transaction metadata information including transaction id, timestamp, transaction fee, transfer list, etc. If a transaction was submitted to multiple nodes, the successful transaction and duplicate transaction(s) will be returned as separate entries in the response with the same transaction ID. Duplicate transactions will still be assessed network fees.

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/transactions"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/transactions/{transactionId}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

consensus timestamp	The consensus timestamp in seconds.nanoseconds
transaction hash	The hash value of the transaction processed on the Hedera network
valid start timestamp	The time the transaction is valid
charged tx fee	The transaction fee that was charged for that transaction
transaction id	The ID of the transaction
memo base64	The memo attached to the transaction encoded in Base64 format
result	Whether the cryptocurrency transaction was successful or not
entity ID	The entity ID that is created from create transactions (AccountCreateTransaction, TopicCreateTransaction, TokenCreateTransaction, ScheduleCreateTransaction, ContractCreateTransaction, FileCreateTransaction).
name	The type of transaction
max fee	The maximum transaction fee the client is willing to pay
valid duration seconds	The seconds for which a submitted transaction is to be deemed valid beyond the start time. The transaction is invalid if consensusTimestamp is greater than transactionValidStart + valid\duration\seconds.
node	The ID of the node that submitted the transaction to the network
transfers	A list of the account IDs the crypto transfer occurred between and the amount that was transferred. A negative (-) sign indicates a debit to that account. The transfer list includes the transfers between the from account and to account, the transfer of the node fee, the transfer of the network fee, and the transfer of the service fee for that transaction. If the transaction was not processed, a network fee is still assessed.
token transfers	The token ID, account, and amount that was transferred to by this account in this transaction. This will not be listed if it did not occur in the transaction
assessed custom fees	The fees that were charged for a custom fee token transfer
links.next	A hyperlink to the next page of responses

Optional Filtering

Operator	Example
Description	

lt (less than)	/api/v1/transactions?
account.id=lt:0.0.1000	
Returns account.id transactions less than 1,000	
lte (less than or equal to)	/api/v1/transactions?
account.id=lte:0.0.1000	
Returns account.id transactions less than or equal to 1,000	
gt (greater than)	/api/v1/transactions?
account.id=gt:0.0.1000	
Returns account.id transactions greater than 1,000	
gte (greater than or equal to)	api/v1/transactions?
account.id=gte:0.0.1000	
Returns account.id transactions greater than or equal to 1,000	
order (order asc or desc values)	
<p><code>/api/v1/transactions?order=asc</code></p><p><code>/api/v1/transactions?order=desc</code></p> <p>Lists transactions in ascending order</p>Lists transactions descending order</p><p></p>	

Note: It is recommended that the account.id query should not select no more than 1000 accounts in a query. If the range specified in the query results in selecting more than 1000 accounts, the API will automatically only search for the first 1000 accounts that match the query in the database and return the transactions for those. For example, if you use ?account.id=gt:0.0.15000 or if you use?account.id=gt:0.0.15000&account.id=lt:0.0.30000, then the API will only return results or some 1000 accounts in this range that match the rest of the query filters.

A single transaction can also be returned by specifying the transaction ID in the request. If a transaction was submitted to multiple nodes, the response will return entries for the successful transaction along with separate entries for the duplicate transaction(s). The "result" key indicates "success" for the node that processed the transaction and "DUPLICATE\TRANSACTION" for each additional node submission. Duplicate entries are still charged network fees.

Parameter	Description

{transactionID}	A specific transaction can be returned by specifying a transaction ID

Additional Examples

Example Request
Description


```

-----
| /api/v1/transactions/?account.id=0.0.1000
| Returns transaction for account ID 1,000
|
| /api/v1/transactions?timestamp=1565779209.711927001
| Returns transactions at 1565779209 seconds and 711927001 nanoseconds
|
| /api/v1/transactions?result=fail
| Returns all transactions that have failed
|
| /api/v1/transactions?account.id=0.0.13622&type=credit /api/v1/transactions?
account.id=0.0.13622&type=debit | <p>Returns all transactions that deposited
into an account ID 0.0.13622</p><p></p><p>Returns all transactions that withdrew
from account ID 0.0.13622<br></p> |
| /api/v1/transactions?transactionType=cryptotransfer
| Returns all cryptotransfer transactions
|

```

Topics

```

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/topics/{topicId}/messages" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

```

Response Details

Response Item	Description
consensus\timestamp	The consensus timestamp of the message in seconds.nanoseconds
topic\id	The ID of the topic the message was submitted to
payer\account\id	The account ID that paid for the transaction to submit the message
message	The content of the message
running\hash	The new running hash of the topic that received the message
sequence\number	The sequence number of the message relative to all other messages for the same topic

```

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/topics/{topicId}/messages/{sequenceNumber}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

```

```

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/topics/messages/{timestamp}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

```

Tokens

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/tokens" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description
token\Id	The ID of the token in x.y.z format
symbol	The symbol of the token
admin\key	The admin key for the token
type	The type of token (fungible or non-fungible)

Additional Examples

Example Request	Description
/api/v1/tokens?publickey=3c3d546321ff6f63d701d2ec5c277095874e19f4a235bee1e6bb19258bf362be	All tokens with matching admin key
/api/v1/tokens?account.id=0.0.8	All tokens for matching account
/api/v1/tokens?token.id=gt:0.0.1001	All tokens in range
/api/v1/tokens?order=desc	All tokens in descending order of token.id
/api/v1/tokens?limit=x	All tokens taking the first x number of tokens

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/tokens/{tokenId}/balances" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description
timestamp	The timestamp of the recorded balances in seconds.nanoseconds
balances	The balance of the tokens in those accounts
account	The ID of the account that has the token balance
balance	The balance of the token associated with the account

Additional Examples

Example Request	Description
/api/v1/tokens/<tokenId>/balances?order=asc	The balance of the token in ascending order
/api/v1/tokens/<tokenId>/balances?account.id=0.0.1000	The balance of the token for account ID 0.0.1000
/api/v1/tokens/<tokenId>/balances?account.balance=gt:1000	The balance of the token associated with the account


```
for the token greater than 1000 |
| /api/v1/tokens/<tokenId>/balances?timestamp=1566562500.040961001 | The token
balances for the specified timestamp |
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/tokens/{tokenId}"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endsswagger %}
```

Response Details

Response Item	Description
-----	-----
admin\key	The token's admin key, if specified
auto\renew\account	The auto renew account ID
auto\renew\period	The period at which the auto renew account will be
charged a renewal fee	
created\timestamp	The timestamp of when the token was created
decimals	The number of decimal places a token is divisible by
expiry\timestamp	The epoch second at which the token should expire
freeze\default	Whether or not accounts created
fee\schedule\key	The fee schedule key, if any
freeze\key	The freeze key for the token, if specified
initial\supply	The initial supply of the token
kyc\key	The KYC key for the token, if specified
modified\timestamp	The last time the token properties were modified
name	The name of the token
supply\key	The supply key for the token, if specified
symbol	The token symbol
token\id	The token ID
total\supply	The total supply of the token
treasury\account\id	The treasury account of the token
type	whether a token is a fungible or non-fungible token
wipe\key	The wipe key for the token, if specified
custom\fees	The custom fee schedule for the token, if any
pause\key	The pause key for a token, if specified
pause\status	Whether or not the token is paused

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/tokens/{tokenId}/nfts" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

account\id	The account ID of the account associated with the NFT
created\timestamp	The timestamp of when the NFT was created
deleted	Whether the token was deleted or not
metadata	The meta data of the NFT
modified\timestamp	The last time the token properties were modified
serial\number	The serial number of the NFT
token\id	The token ID of the NFT

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/tokens/{tokenId}/nfts/{serialNumber}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

account\id	The account ID of the account associated with the NFT
created\timestamp	The timestamp of when the NFT was created
deleted	Whether the token was deleted or not
metadata	The meta data of the NFT
modified\timestamp	The last time the token properties were modified
serial\number	The serial number of the NFT
token\id	The token ID of the NFT

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/tokens/{tokenId}/nfts/{serialNumber}/transactions" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

created\timestamp	The timestamp of the transaction
id	The timestamp of the transaction
receiver\account\id	The account that received the NFT
sender\account\id	The account that sent the NFT
type	The type of transaction
token\id	The token ID of the NFT

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/supply" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
```

```
mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Schedule Transactions

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/schedules"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description
-----	-----
schedules	List of schedules
admin\key	The admin key on the schedule
admin\key\type	The type of key
admin\key\key	The admin public key
consensus\timestamp	The consensus timestamp of when the schedule was created
creator\account\id	The account ID of the creator of the schedule
executed\timestamp	The timestamp at which the transaction that was scheduled was executed at
memo	A string of characters associated with the memo if set
payer\account\id	The account ID of the account paying for the execution of the transaction
schedule\id	The ID of the schedule entity
signatures	The list of keys that signed the transaction
signatures.public\key\prefix	The signatures public key prefix
signatures.signature	The signature of the key that signed the schedule transaction
signatures.type	The type of signature (ED5519 or ECDSA)
transaction\body	The transaction body of the transaction that was scheduled
wait\for\expiry	Whether or not the schedule transaction specified a specific time to expire by
links.next	Hyperlink to the next page of results

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/schedules/{scheduleId}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Details

Response Item	Description

adminKey	The admin key on the schedule
adminKey.type	The type of key
adminKey.key	The admin public key
consensus.timestamp	The consensus timestamp of when the schedule was created
creator.account.id	The account ID of the creator of the schedule
executed.timestamp	The timestamp at which the transaction that was scheduled was executed at
memo	A string of characters associated with the memo if set
payer.account.id	The account ID of the account paying for the execution of the transaction
schedule.id	The ID of the schedule entity
signatures	The list of keys that signed the transaction
signatures.consensus.timestamp	The consensus timestamp at which the signature was added
signatures.public.key.prefix	The signatures public key prefix
signatures.signature	The signature of the key that signed the schedule transaction
signatures.type	The type of signature (ED5519 or ECDSA)
transaction.body	The transaction body of the transaction that was scheduled
wait.for.expiry	Whether or not the schedule transaction specified a specific time to expire by

Smart Contracts

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/contracts"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

Response Item	Description

admin.key	The admin key of the contract, if specified
auto.renew.account	The account paying the auto renew fees, if set
auto.renew.period	The period at which the contract auto renews
bytecode	The bytecode of the contract
contract.id	The contract ID
created.timestamp	The timestamp the contract was created at
deleted	Whether or not the contract is deleted
evm.address	The EVM address of the contract

expiration\timestamp		The timestamp of when the contract is set
to expire		
file\id		The ID of the file that stored the
contract bytecode		
max\automatic\token\associations		The number of automatic token association
slots		
memo		The memo of the contract, if specified
obtainer\id		The ID of the account or contract that
will receive any remaining balance when the contract is deleted		
permanent\removal		Set to true when the system expires a
contract		
proxy\account\id		The proxy account ID (disabled)
solidity\address		The solidity address
timestamp		The period for which the attributes are
valid for		

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/{contractIdOrAddress}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/{contractIdOrAddress}/results" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/{contractIdOrAddress}/state" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/{contractIdOrAddress}/results/{timestamp}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/contracts/results"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/results/{transactionIdOrHash}" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
```

```

main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/results/{transactionIdOrHash}/actions" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/results/{transactionIdOrHash}/opcodes" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/{contractIdOrAddress}/results/logs" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml"
path="/api/v1/contracts/results/logs" method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/contracts/call"
method="post" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

```

Blocks

```

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/blocks" method="get"
%}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/blocks/{hashOrNumber}"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

```

Network

```

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/supply"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml
{% endswagger %}

{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/
main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/fees"
method="get" %}
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-
mirror-rest/api/v1/openapi.yml

```

```
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/exchangerate" method="get" %}  
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml  
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/nodes" method="get" %}  
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml  
{% endswagger %}
```

```
{% swagger src="https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml" path="/api/v1/network/stake" method="get" %}  
https://raw.githubusercontent.com/hashgraph/hedera-mirror-node/main/hedera-mirror-rest/api/v1/openapi.yml  
{% endswagger %}
```

```
# README.md:
```

Hedera APIs

```
{% hint style="danger" %}
```

This section is currently not up to date and will be updated with the auto-generated protobuf docs here. Please reference the SDKs section to learn about our APIs.

```
{% endhint %}
```

```
# accountamount.md:
```

AccountAmount

An account, and the amount that it sends or receives during a cryptocurrency or token transfer.

Field	Type	Description
-----	-----	-----
accountID	AccountID	The Account ID that sends/receives cryptocurrency or tokens
amount	sint64	The amount of tinybars (for Crypto transfers) or in the lowest denomination (for Token transfers) that the account sends(negative) or receives(positive)
isapproval	bool	If true then the transfer is expected to be an approved allowance and the accountID is expected to be the owner. The default is false (omitted).

```
####
```

```
# accountid.md:
```

AccountID

The ID for a cryptocurrency account.

Field	Type	Description
-----	-----	
shardNum	int64	The shard number (nonnegative)
realmNum	int64	The realm number (nonnegative)
accountNum	int64	A nonnegative account number unique within its realm
alias	bytes	The public key bytes to be used as the account's alias. The public key bytes are the result of serializing a protobuf Key message for any primitive key type. Currently only primitive key bytes are supported as an alias (ThresholdKey, KeyList, ContractID, and delegatable\contract\id are not supported). At most one account can ever have a given alias and it is used for account creation if it was automatically created using a crypto transfer. It will be null if an account is created normally. It is immutable once it is set for an account. If a transaction auto-creates the account, any further transfers to that alias will simply be deposited in that account, without creating anything, and with no creation fee being charged.

contractid.md:

ContractID

The ID for a smart contract instance

Field	Description

shardNum	The shard number (nonnegative)
realmNum	The realm number (nonnegative)
contractNum	A nonnegative number unique within its realm
evmaddress	<p>The 20-byte EVM address of the contract to call. Every contract has an EVM address determined by its shard.realm.num id. This address is as follows:</p> <p>The first 4 bytes are the big-endian representation of the shard.</p> <p>The next 8 bytes are the big-endian representation of the realm.</p> <p>The final 8 bytes are the big-endian representation of the number.</p> <p>Contracts created via CREATE2 have an additional, primary</p>

address

 that is derived from the [EIP-1014](https://eips.ethereum.org/EIPS/eip-1014) specification, and does not have a simple relation to a shard.realm.num id. (Please do note that CREATE2 contracts can also be referenced by the three-part EVM address described above.)

####

cryptoallowance.md:

CryptoAllowance

An approved allowance of hbar transfers for a spender.

Field	Type	Description
owner	AccountID	The account ID of the hbar owner (ie. the grantor of the allowance)
spender	AccountID	The account ID of the spender of the hbar allowance
amount	int64	The amount of the spender's allowance in tinybars

####

currentandnextfeeschedule.md:

CurrentAndNextFeeSchedule

This contains two Fee Schedules with expiry timestamp.

Field	Type	Description
currentFeeSchedule	FeeSchedule	Contains current Fee Schedule
nextFeeSchedule	FeeSchedule	Contains next Fee Schedule

####

feecomponents.md:

FeeComponents

The different components used for fee calculation

Field	Description
min	The minimum fees that needs to be paid
max	The maximum fees that can be submitted
constant	A constant determined by the business to calculate the fees
bpt	Bytes per transaction

vpt	Verifications per transaction
rbh	Ram byte seconds
sbh	Storage byte seconds
gas	Gas for the contract execution
tv	Transaction value (crypto transfers amount, tv is in tiny bars
divided by 1000, rounded down)	
bpr	Bytes per response
sbpr	Storage bytes per response

####

feedata.md:

FeeData

The total fees charged for a transaction. It contains three parts namely node data, network data and service data

Field	Type	Description
-----	-----	

nodedata	FeeComponents	Fee charged by Node for this functionality
networkdata	FeeComponents	Fee charged for network operations by Hedera
servicedata	FeeComponents	Fee charged for providing service by Hedera
subType	SubType	SubType distinguishing between different
types of FeeData, correlating to the same HederaFunctionality		

####

feeschedule.md:

FeeSchedule

The fee schedule for a specific hedera functionality and the time period this fee schedule will expire

Field	Type
Description	

transactionFeeSchedule	TransactionFeeSchedule
multiple functionality specific fee schedule.	Contains
expiryTime	TimestampSeconds FeeSchedule expiry time

####

fileid.md:

FileID

The ID for a file

Field	Description
shardNum	The shard number (nonnegative)
realmNum	The realm number (nonnegative)
fileNum	A nonnegative File number unique within its realm

####

fraction.md:

Fraction

A rational number, used to set the amount of a value transfer to collect as a custom fee.

Field	Type	Description
numerator	int64	The rational's numerator
denominator	int64	The rational's denominator; a zero value will result in FRACTION\DIVIDES\BY\ZERO

hederafunctionality.md:

HederaFunctionality

The functionality provided by Hedera.

Enum Name	Description
NONE	UNSPECIFIED - Need to keep first value as unspecified because first element is ignored and not parsed (0 is ignored by parser)
CryptoTransfer	Crypto transfer
CryptoUpdate	Crypto update account
CryptoDelete	Crypto delete account
CryptoAddLiveHash	Add a livehash to a crypto account (0.5.0)
CryptoDeleteLiveHash	Delete a livehash from a crypto account (0.5.0)
CryptoAddClaim	Crypto add claim to the account
CryptoDeleteClaim	Crypto delete claim to the account
ContractCall	Smart Contract Call
ContractCreate	Smart Contract Create Contract
ContractUpdate	Smart Contract update contract

FileCreate	File Operation create file
FileAppend	File Operation append file
FileUpdate	File Operation update file
FileDelete	File Operation delete file
CryptoGetAccountBalance	Crypto get account balance
CryptoGetAccountRecords	Crypto get account record
CryptoGetInfo	Crypto get info
ContractCallLocal	Smart Contract Call
ContractGetInfo	Smart Contract get info
ContractGetBytecode	Smart Contract, get the byte code
GetBySolidityID	Smart Contract, get by solidity ID
GetByKey	Smart Contract, get by key
CryptoGetClaim	Crypto get the claim
CryptoGetLiveHash	Get a live hash from a crypto account
CryptoGetStakers	Crypto, get the stakers for the node
FileGetContents	File Operations get file contents
FileGetInfo	File Operations get the info of the file
TransactionGetRecord	Crypto get the transaction records
ContractGetRecords	Contract get the transaction records
CryptoCreate	crypto create account
SystemDelete	System delete file
SystemUndelete	System undelete file
ContractDelete	Delete contract
Freeze	Freeze
CreateTransactionRecord	Create Tx Record
CryptoAccountAutoRenew	Crypto Auto Renew
ContractAutoRenew	Contract Auto Renew
getVersion	Get Version
TransactionGetReceipt	Transaction Get Receipt
ConsensusCreateTopic	Create a topic
ConsensusUpdateTopic	Update a topic
ConsensusDeleteTopic	Delete a topic

ConsensusGetTopicInfo	Get topic info
ConsensusSubmitMessage	Submit a message to a topic
TokenCreate	Create Token
TokenTransact	Transfer Tokens
TokenGetInfo	Transfer Tokens
TokenFreezeAccount	Freeze Account
TokenUnfreezeAccount	Unfreeze Account
TokenGrantKycToAccount	Grant KYC to Account
TokenRevokeKycFromAccount	Revoke KYC from Account
TokenDelete	Delete Token
TokenUpdate	Update Token
TokenMint	Mint tokens to treasury
TokenBurn	Burn tokens from treasury
TokenAccountWipe	Wipe token amount from Account holder
TokenAssociateToAccount	Wipe token amount from Account holder
TokenDissociateFromAccount	Dissociate tokens from an account
ScheduleCreate	Create Scheduled Transaction
ScheduleDelete	Delete Scheduled Transaction
ScheduleSign	Sign Scheduled Transaction
ScheduleGetInfo	Get Scheduled Transaction Information
TokenGetAccountNftInfo	Get Token Account Nft Information
TokenGetNftInfo	Get Token Nft Information
TokenGetNftInfos	Get Token Nft List Information
NetworkGetExecutionTime available	Get execution time(s) by TransactionID, if available
TokenPause	Pause the Token
TokenUnpause	Unpause the Token
CryptoApproveAllowance payer account	Approve allowance for a spender relative to the payer account
CryptoDeleteAllowance account	Deletes granted NFT allowances on an owner account

key.md:

Key

A Key can be a public key from either the Ed25519 or ECDSA(secp256k1) signature schemes, where in the ECDSA(secp256k1) case we require the 33-byte compressed form of the public key. We call these public keys primitive keys.

If an account has primitive key associated to it, then the corresponding private key must sign any transaction to transfer cryptocurrency out of it. A Key can also be the ID of a smart contract instance, which is then authorized to perform any precompiled contract action that requires this key to sign.

Note that when a Key is a smart contract ID, it doesn't mean the contract with that ID will actually create a cryptographic signature. It only means that when the contract calls a precompiled contract, the resulting "child transaction" will be authorized to perform any action controlled by the Key.

A Key can be a "threshold key", which means a list of M keys, any N of which must sign in order for the threshold signature to be considered valid. The keys within a threshold signature may themselves be threshold signatures, to allow complex signature requirements.

A Key can be a "key list" where all keys in the list must sign unless specified otherwise in the documentation for a specific transaction type (e.g. FileDeleteTransactionBody). Their use is dependent on context. For example, a Hedera file is created with a list of keys, where all of them must sign a transaction to create or modify the file, but only one of them is needed to sign a transaction to delete the file. So it's a single list that sometimes acts as a 1-of-M threshold key, and sometimes acts as an M-of-M threshold key. A key list is always an M-of-M, unless specified otherwise in documentation. A key list can have nested key lists or threshold keys. Nested key lists are always M-of-M. A key list can have repeated primitive public keys, but all repeated keys are only required to sign once.

A Key can contain a ThresholdKey or KeyList, which in turn contain a Key, so this mutual recursion would allow nesting arbitrarily deep. A ThresholdKey which contains a list of primitive keys has 3 levels: ThresholdKey -> KeyList -> Key. A KeyList which contains several primitivekeys has 2 levels: KeyList -> Key. A Key with 2 levels of nested ThresholdKeys has 7 levels: Key -> ThresholdKey -> KeyList -> Key -> ThresholdKey -> KeyList -> Key.

Each Key should not have more than 46 levels, which implies 15 levels of nested ThresholdKeys.

Field	Type	Description
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
key	oneof	
contractID	ContractID	smart contract instance that is authorized as if it had signed with a key
ed25519 public key bytes		ed25519
RSA3072 public key bytes (NOT SUPPORTED)		RSA-3072
ECDSA384 the p-384 curve public key bytes (NOT SUPPORTED)		ECDSA with

thresholdKey	ThresholdKey	a threshold N followed by a list of M keys, any N of which are required to form a valid signature
keyList	KeyList	A list of Keys of the Key type.
ECDSAsecp256k1		Compressed ECDSA(secp256k1) public key bytes
delegatablecontract-id	ContractID	A smart contract that, if the recipient of the active message frame, should be treated as having signed. (Note this does not mean the code being executed in the frame will belong to the given contract, since it could be running another contract's code via delegatecall. So setting this key is a more permissive version of setting the contractID key, which also requires the code in the active message frame belong to the contract with the given id.)

\\

keylist.md:

KeyList

A list of keys that requires all keys (M-of-M) to sign unless otherwise specified in the documentation. A KeyList may contain repeated keys, but all repeated keys are only required to sign once.

Field	Type	Description
keys	Key	list of keys

nftallowance.md:

NftAllowance

An approved allowance of NFT transfers for a spender.

Field	Type	Description
tokenId	TokenID	The token that the allowance pertains to
owner	AccountID	The account ID of the hbar owner (ie. the grantor of the allowance)
spender	AccountID	The account ID of the spender of the hbar allowance
serialNumbers	repeated int64	The list of serial numbers that the spender is permitted to transfer
approvedForAll	google.protobuf.BoolValue	If true, the spender has access to all of the account owner's NFT instances (currently \\ owned and any in the future). If this field is set to true the

serialNumbers field should be empty. |
 | delegatingspender | AccountID | The
 account ID of the spender who is granted approvedForAll allowance and granting
 approval on an NFT serial to another spender.
 |

####

nfttransfer.md:

NftTransfer

A sender account, a receiver account, and the serial number of an NFT of a Token
 with NONFUNGIBLEUNIQUE type.

Field	Type	Description
-----	-----	-----
senderAccountID	AccountID	The accountID of the sender
receiverAccountID	AccountID	The accountID of the receiver
serialNumber	int64	The serial number of the NFT
isapproval	bool	If true then the transfer is expected to be an approved allowance and the senderAccountID is expected to be the owner. The default is false (omitted).

nodeaddress.md:

NodeAddress

The metadata for a Node - including IP Address, and the crypto account
 associated with the Node.

Field	Type	Description
-----	-----	-----
ipAddress	bytes	The ip address of the Node with separator & octets \[deprecated v0.13.0]
portno	int32	The port number of the grpc server for the node \[deprecated v0.13.0]
memo	bytes	The memo field of the node \[deprecated v0.13.0]
RSAPubKey	string	The RSA public key of the node.
nodeId	int64	A non- sequential identifier for the node
nodeAccountId	AccountID	The account to be paid for queries and transactions sent to this node
nodeCertHash	bytes	A hash of the X509 cert used for gRPC traffic to this node
serviceEndpoint	repeated ServiceEndpoint	A node's service IP addresses and ports \[Added v0.13.0]
description	string	A description of the node, with UTF-8 encoding up to 100 bytes \[Added v0.13.0]
stake	int64	The amount

of tinybars staked to the node \[Added v0.13.0]

|

nodeaddressbook.md:

NodeAddressBook

A list of nodes and their metadata.

Field	Type	Description
-----	-----	
nodeAddress	NodeAddress	Contains multiple Node Address for the network

README.md:

Basic Types

realmid.md:

RealmID

The ID for a realm. Within a given shard, every realm has a unique ID. Each account, file, and contract instance belongs to exactly one realm.

Field	Description
-----	-----
shardNum	The shard number (nonnegative)
realmNum	The realm number (nonnegative)

\
\

scheduleid.md:

ScheduleID

A unique identifier for a Schedule.

Field	Type	Description
-----	-----	-----
shardNum	int64	The shard number (nonnegative)
realmNum	int64	The realm number (nonnegative)
scheduleNum	int64	A nonnegative schedule number

\
\

semanticversion.md:

SemanticVersion

Hedera follows semantic versioning (<https://semver.org/>) for both the HAPI protobufs and the Services software. This type allows the getVersionInfo query in the NetworkService to return the deployed versions of both protobufs and software on the node answering the query.

Field	Type	Description
-------	------	-------------

Field	Type	Description
major	int32	Increases with incompatible API changes
minor	int32	Increases with backwards-compatible new functionality
patch	int32	Increases with backwards-compatible bug fixes
pre	string	A pre-release version MAY be denoted by appending a hyphen and a series of dot separated identifiers (https://semver.org/#spec-item-9); so given a semver 0.14.0-alpha.1+21AF26D3, this field would contain 'alpha.1'
build	string	Build metadata MAY be denoted by appending a plus sign and a series of dot separated identifiers immediately following the patch or pre-release version (https://semver.org/#spec-item-10); so given a semver 0.14.0-alpha.1+21AF26D3, this field would contain '21AF26D3'

####

serviceendpoint.md:

ServiceEndpoint

Contains the IP address and the port representing a service endpoint of a Node in a network. Used to reach the Hedera API and submit transactions to the network.

Field	Type	Description
Hedera Services Version		
ipAddressV4	bytes	The 32-bit IPv4 address of the node encoded in left to right order (e.g. 127.0.0.1 has 127 as its first byte)
port	int32	The port of the node

servicesconfigurationlist.md:

ServicesConfigurationList

Field	Type	Description
nameValue	Setting	list of name value pairs of the application properties

setting.md:

Setting

Field	Description
name	name of the property
value	value of the property
data	any data associated with property

shardid.md:

ShardID

Each shard has a nonnegative shard number. Each realm within a given shard has a nonnegative realm number (that number might be reused in other shards). And each account, file, and smart contract instance within a given realm has a nonnegative number (which might be reused in other realms). Every account, file, and smart contract instance is within exactly one realm. So a FileID is a triplet of numbers, like 0.1.2 for entity number 2 within realm 1 within shard 0. Each realm maintains a single counter for assigning numbers, so if there is a file with ID 0.1.2, then there won't be an account or smart contract instance with ID 0.1.2.

Field	Description
shardNum	the shard number (nonnegative)

signature-list.md:

SignatureList

{% hint style="info" %}

This message is deprecated and succeeded by SignaturePair and SignatureMap messages.

{% endhint %}

The signatures corresponding to a KeyList of the same length.

Field	Type	Description
option	deprecated=true	
sigs	Signature	each signature corresponds to a Key in the KeyList

signature-pair.md:

SignaturePair

The client may use any number of bytes from zero to the whole length of the public key for pubKeyPrefix. If zero bytes are used, then it must be that only one primitive key is required to sign the linked transaction; it will surely resolve to INVALID\SIGNATURE otherwise.

{% hint style="info" %}

IMPORTANT: In the special case that a signature is being provided for a key used to authorize a precompiled contract, the pubKeyPrefix must contain the entire public key! That is, if the key is a Ed25519 key, the pubKeyPrefix should be 32 bytes long. If the key is a ECDSA(secp256k1) key, the pubKeyPrefix should be 33 bytes long, since we require the compressed form of the public key.

{% endhint %}

Only Ed25519 and ECDSA(secp256k1) keys and hence signatures are currently supported.

Field	Type	Description
-------	------	-------------

pubKeyPrefix		First few bytes of the public key	
signature	oneof		
contract virtual signature	contract signature (always length zero)		smart
signature	ed25519		ed25519
signature	RSA\3072		RSA-3072
signature	ECDSA\384		ECDSA p-
384 signature			
ECDSA(secp256k1) signature	ECDSA\secp256k1		
\			
\\			

signature.md:

Signature

```
{% hint style="info" %}
```

This message is deprecated and succeeded by SignaturePair and SignatureMap messages.

```
{% endhint %}
```

A Signature corresponding to a Key. It is a sequence of bytes holding a public key signature from one of the three supported systems (ed25519, RSA-3072, ECDSA with p384). Or, it can be a list of signatures corresponding to a single threshold key. Or, it can be the ID of a smart contract instance, which is authorized to act as if it had a key. If an account has an ed25519 key associated with it, then the corresponding private key must sign any transaction to transfer cryptocurrency out of it. If it has a smart contract ID associated with it, then that smart contract is allowed to transfer cryptocurrency out of it. The smart contract doesn't actually have a key, and doesn't actually sign a transaction. But it's as if a virtual transaction were created, and the smart contract signed it with a private key. A key can also be a "threshold key", which means a list of M keys, any N of which must sign in order for the threshold signature to be considered valid. The keys within a threshold signature may themselves be threshold signatures, to allow complex signature requirements (this nesting is not supported in the currently, but will be supported in a future version of API). If a Signature message is missing the "signature" field, then this is considered to be a null signature. That is useful in cases such as threshold signatures, where some of the signatures can be null.

Field	Type	Description
-------	------	-------------

deprecated		
signature	oneof	

	contract	
Smart contract virtual signature (always length zero)		
	ed25519	
Ed25519 signature bytes		
	RSA\3072	
RSA-3072 signature bytes		
	ECDSA\384	
ECDSA p-384 signature bytes		
	thresholdSignature	ThresholdSignature
A list of signatures for a single N-of-M threshold Key. This must be a list of exactly M signatures, at least N of which are non-null.		
	signatureList	SignatureList
A list of M signatures, each corresponding to a Key in a KeyList of the same length.		

signaturemap.md:

SignatureMap

A set of signatures corresponding to every unique public key used to sign a given transaction.

Field	Type	Description
-----	-----	-----
-		
sigPair	SignaturePair	Each signature pair corresponds to a unique Key required to sign the transaction.

subtype.md:

SubType

Possible FeeData Object SubTypes. Supplementary to the main HederaFunctionality Type. When not explicitly specified, DEFAULT is used.

Enum Name	Description
-----	-----
DEFAULT	The resource prices have no special scope
TOKENFUNGIBLECOMMON	The resource prices are scoped to an operation on a fungible common token
TOKENNONFUNGIBLEUNIQUE	The resource prices are scoped to an operation on a non-fungible unique token
TOKENFUNGIBLECOMMONWITHCUSTOMFEES	The resource prices are scoped to an operation on a fungible common token with a custom fee schedule
TOKENNONFUNGIBLEUNIQUEWITHCUSTOMFEES	The resource prices are scoped to an operation on a non-fungible unique token with a custom fee schedule

thresholdkey.md:

ThresholdKey

A set of public keys that are used together to form a threshold signature. If the threshold is N and there are M keys, then this is an N of M threshold signature. If an account is associated with ThresholdKeys, then a transaction to move cryptocurrency out of it must be signed by a list of M signatures, where at most M-N of them are blank, and the other at least N of them are valid signatures corresponding to at least N of the public keys listed here.

Field	Type	Description
threshold	int	A valid signature set must have at least this many signatures
keys	KeyList	List of all the keys that can sign

thresholdsignature.md:

ThresholdSignature

{% hint style="info" %}

This message is deprecated and succeeded by SignaturePair and SignatureMap messages.

{% endhint %}

A signature corresponding to a ThresholdKey. For an N-of-M threshold key, this is a list of M signatures, at least N of which must be non-null.

Field	Type	Description
option	deprecated=true	
signatures	SignatureList	for an N-of-M threshold key, this is a list of M signatures, at least N of which must be non-null

tokenallowance.md:

TokenAllowance

An approved allowance of token transfers for a spender.

Field	Type	Description
tokenId	TokenID	The token that the allowance pertains to
owner	AccountID	The account ID of the hbar owner (ie. the grantor of the allowance)
spender	AccountID	The account ID of the spender of the hbar allowance
amount	int64	The amount of the spender's token allowance

####

tokenbalance.md:

TokenBalance

A number of transferable units of a certain token. The transferable unit of a token is its smallest denomination, as given by the token's decimals property---each minted token contains 10decimals transferable units. For example, we could think of the cent as the transferable unit of the US dollar (decimals=2); and the tinybar as the transferable unit of hbar (decimals=8). Transferable units are not directly comparable across different tokens.

Field	Type	Description
-----	-----	

tokenId	TokenID	A unique token id
balance	uint64	A number of transferable units of the identified token. For token of type FUNGIBLECOMMON - balance in the smallest denomination. For token of type NONFUNGIBLEUNIQUE - the number of NFTs held by the account
decimals	uint32	Tokens divide into 10 decimals pieces

tokenbalances.md:

TokenBalances

A sequence of token balances.

Field	Type	Description
-----	-----	

tokenBalances	repeated TokenBalances	A sequence of token balances

tokenfreezestatus.md:

TokenFreezeStatus

Possible Freeze statuses returned on TokenGetInfoQuery or CryptoGetInfoResponse in TokenRelationship

Enum Name	Description
-----	-----
FreezeNotApplicable	Freeze n/a
Frozen	Token cannot be transferred to the account
Unfrozen	Token can be transferred to the account

tokenId.md:

TokenID

A unique identifier for a token.

Field	Description
-----	-----

shardNum	A nonnegative shard number
realmNum	A nonnegative realm number
tokenNum	A nonnegative token number

####

tokenkycstatus.md:

TokenKycStatus

Possible KYC statuses returned on TokenGetInfoQuery or CryptoGetInfoResponse in TokenRelationship.

Enum Name	Description
KycNotApplicable	KYC n/a
Granted	KYC was granted
Revoked	KYC was revoked

####

tokenpausestatus.md:

TokenPauseStatus

Possible Pause statuses returned on TokenGetInfoQuery.

Enum Name	Description
PauseNotApplicable	Indicates that a Token has no pauseKey
Paused	Indicates that a Token is Paused
Unpaused	Indicates that a Token is Unpaused

tokenrelationship.md:

TokenRelationship

Token's information related to the given Account.

Field	Type	Description
tokenId	TokenID	A unique token id
symbol	string	The Symbol of the token
balance	uint64	For token of type FUNGIBLECOMMON - the balance that the Account holds in the smallest denomination. For token of type NONFUNGIBLEUNIQUE - the number of NFTs held by the account
kycStatus	TokenKycStatus	The KYC status of the account (KycNotApplicable, Granted or Revoked). If the token does not have KYC key, KycNotApplicable is returned
freezeStatus	TokenFreezeStatus	The Freeze status of the account (FreezeNotApplicable, Frozen or Unfrozen). If the token does not have Freeze key, FreezeNotApplicable is returned

decimals	uint32	Tokens
divide into 10 decimal pieces		
automaticassociation	bool	

Specifies if the relationship is created implicitly.
False : explicitly associated,
True : implicitly associated.

tokensupplytype.md:

TokenSupplyType

Possible Token Supply Types (IWA Compatibility). Indicates how many tokens can have during its lifetime.

Enum Name	Description
INFINITE	Indicates that tokens of that type have an upper bound of Long.MAX\VALUE.
FINITE	Indicates that tokens of that type have an upper bound of maxSupply, provided on token creation.

tokentransferlist.md:

TokenTransferList

A list of token IDs and amounts representing the transferred out (negative) or into (positive) amounts, represented in the lowest denomination of the token.

Field	Type	Description
token	TokenID	The ID of the token
transfers	repeated AccountAmount	Multiple list of AccountAmounts, each of which has an account and amount
nftTransfers	repeated NftTransfers	Applicable to tokens of type NON\FUNGIBLE\UNIQUE. Multiple list of NftTransfers, each of which has a sender and receiver account, including the serial number of the NFT
expecteddecimals	google.protobuf.UInt32Value	If present, the number of decimals this fungible token type is expected to have. The transfer will fail with UNEXPECTED\TOKEN\DECIMALS if the actual decimals differ.

tokentype.md:

TokenType

Possible Token Types (IWA Compatibility).\

Apart from fungible and non-fungible, Tokens can have either a common or unique representation. This distinction might seem subtle, but it is important when considering how tokens can be traced and if they can have isolated and unique properties.

Enum Name	Description

FUNGIBLECOMMON	Interchangeable value with one another, where any quantity of them has the same value as another equal quantity if they are in the same class. Share a single set of properties, not distinct from one another. Simply represented as a balance or quantity to a given Hedera account.
NONFUNGIBLEUNIQUE	Unique, not interchangeable with other tokens of the same type as they typically have different values. Individually traced and can carry unique properties (e.g. serial number).

####

topicid.md:

TopicID

Unique identifier for a topic (used by the consensus service)

Field	Description
-----	-----
shardNum	The shard number (nonnegative)
realmNum	The realm number (nonnegative)
topicNum	Unique topic identifier within a realm (nonnegative).

transactionfeeschedule.md:

TransactionFeeSchedule

The fees for a specific transaction or query based on the fee data.

Field	Type
Description	
-----	-----
hederaFunctionality	HederaFunctionality A particular transaction or query
feeData	FeeData <p>Resource price
coefficients</p><p>[deprecated]</p>	
fees	repeated FeeData Resource price
coefficients. Supports subtype price definition	

transactionid.md:

TransactionID

The ID for a transaction. This is used for retrieving receipts and records for a transaction, for appending to a file right after creating it, for instantiating a smart contract with bytecode in a file just created, and internally by the network for detecting when duplicate transactions are submitted. A user might get a transaction processed faster by submitting it to N nodes, each with a

different node account, but all with the same TransactionID. Then, the transaction will take effect when the first of all those nodes submits the transaction and it reaches consensus. The other transactions will not take effect. So this could make the transaction take effect faster, if any given node might be slow. However, the full transaction fee is charged for each transaction, so the total fee is N times as much if the transaction is sent to N nodes.

Applicable to Scheduled Transactions

The ID of a Scheduled Transaction has transactionValidStart and accountIDs inherited from the ScheduleCreate transaction that created it. That is to say that they are equal

The scheduled property is true for Scheduled Transactions

transactionValidStart, accountID and scheduled properties should be omitted

Field	Type	Description
transactionValidStart	Timestamp	The transaction is invalid if consensusTimestamp < transactionValidStart
accountID	AccountID	The Account ID that paid for this transaction
scheduled	bool	Whether the Transaction is of type Scheduled or no
nonce	int32	The identifier for an internal transaction that was spawned as part of handling a user transaction. (These internal transactions share the transactionValidStart and accountID of the user transaction, so a nonce is necessary to give them a unique TransactionID.) An example is when a "parent" ContractCreate or ContractCall transaction calls one or more HTS precompiled contracts; each of the "child" transactions spawned for a precompile has a id with a different nonce.

transferlist.md:

TransferList

A list of accounts and amounts to transfer out of each account (negative) or into it (positive).

Field	Type	Description
accountAmounts	repeated AccountAmount	Multiple list of AccountAmount pairs, each of which has an account and an amount to transfer into it (positive) or out of it (negative)

consensus-service.md:

Consensus Service

The Consensus Service provides the ability for Hedera to provide aBFT consensus as to the order and validity of messages submitted to a topic, as well as a consensus timestamp for those messages.\

\ Automatic renewal can be configured via an autoRenewAccount (not currently implemented).\

Any time an autoRenewAccount is added to a topic, that createTopic/updateTopic transaction must be signed by the autoRenewAccount.\

\ The autoRenewPeriod on an account must currently be set a value in createTopic between MIN\AUTORENEW\PERIOD (6999999seconds) and MAX\AUTORENEW\PERIOD (8000001 seconds). During creation this sets the initial expirationTime of the topic (see more below).\

\ If no adminKey is on a topic, there may not be an autoRenewAccount on the topic, deleteTopic is not allowed, and the only change allowed via an updateTopic is to extend the expirationTime.\

\ If an adminKey is on a topic, every updateTopic and deleteTopic transaction must be signed by the adminKey, except for updateTopics which only extend the topic's expirationTime (no adminKey authorization required).\

\ If an updateTopic modifies the adminKey of a topic, the transaction signatures on the updateTopic must fulfill both the pre-update and post-update adminKey signature requirements.\

\ Mirrornet ConsensusService may be used to subscribe to changes on the topic, including changes to the topic definition and the consensus ordering and timestamp of submitted messages.\

\ Until autoRenew functionality is supported by HAPI, the topic will not expire, the autoRenewAccount will not be charged, and the topic will not automatically be deleted.

\ Once autoRenew functionality is supported by HAPI:

1. Once the expirationTime is encountered, if an autoRenewAccount is configured on the topic, the account will be charged automatically at the expirationTime, to extend the expirationTime of the topic up to the topic's autoRenewPeriod (or as much extension as the account's balance will supply).
2. If the topic expires and is not automatically renewed, the topic will enter the EXPIRED state. All transactions on the topic will fail with TOPIC\EXPIRED, except an updateTopic() call that modifies only the expirationTime. getTopicInfo() will succeed. This state will be available for a AUTORENEW\GRACE\PERIOD grace period (7 days).
3. After the grace period, if the topic's expirationTime is not extended, the topic will be automatically deleted and no transactions or queries on the topic will succeed after that point.

ConsensusService

RPC	Request	Response
Comments		
-----	-----	
-----	-----	
-----	-----	
-----	-----	
-----	-----	

```

-----
-----
-----
|
| createTopic | Transaction | TransactionResponse | <p>Create a topic to be
used for consensus.<br>If an autoRenewAccount is specified, that account must
also sign this transaction.<br>If an adminKey is specified, the adminKey must
sign the transaction.<br>On success, the resulting TransactionReceipt contains
the newly created TopicId.<br>Request is <a
href="https://github.com/theekrystallee/hedera-style-guide/blob/sdk-v1/
deprecated/hedera-api/consensus-service/broken-reference/
README.md">ConsensusCreateTopicTransactionBody</a></p>
|
| updateTopic | Transaction | TransactionResponse | <p>Update a topic.<br>If
there is no adminKey, the only authorized update (available to anyone) is to
extend the expirationTime.<br>Otherwise transaction must be signed by the
adminKey.<br>If an adminKey is updated, the transaction must be signed by the
pre-update adminKey and post-update adminKey.<br>If a new autoRenewAccount is
specified (not just being removed), that account must also sign the
transaction.<br>Request is <a href="https://github.com/theekrystallee/hedera-
style-guide/blob/sdk-v1/deprecated/hedera-api/consensus-service/broken-
reference/README.md">ConsensusUpdateTopicTransactionBody</a></p>
|
| deleteTopic | Transaction | TransactionResponse | <p>Delete a topic. No more
transactions or queries on the topic (via HAPI) will succeed.<br>If an adminKey
is set, this transaction must be signed by that key.<br>If there is no adminKey,
this transaction will fail UNAUTHORIZED.<br>Request is <a
href="https://github.com/theekrystallee/hedera-style-guide/blob/sdk-v1/
deprecated/hedera-api/consensus-service/broken-reference/
README.md">ConsensusDeleteTopicTransactionBody</a></p>
|
| getTopicInfo | Query | Response |
<p>Retrieve the latest state of a topic. This method is unrestricted and allowed
on any topic by any payer account.<br>Deleted accounts will not be
returned.<br>Request is <a href="https://github.com/theekrystallee/hedera-style-
guide/blob/sdk-v1/deprecated/hedera-api/consensus-service/broken-reference/
README.md">ConsensusGetTopicInfoQuery</a><br>Response is <a
href="https://github.com/theekrystallee/hedera-style-guide/blob/sdk-v1/
deprecated/hedera-api/consensus-service/broken-reference/
README.md">ConsensusGetTopicInfoResponse</a></p>
|
| submitMessage | Transaction | TransactionResponse | <p>Submit a message for
consensus.<br>Valid and authorized messages on valid topics will be ordered by
the consensus service, gossipped to the<br>mirror net, and published (in order)
to all subscribers (from the mirror net) on this topic.<br>The submitKey (if
any) must sign this transaction.<br>On success, the resulting TransactionReceipt
contains the topic's updated topicSequenceNumber
and<br>topicRunningHash.<br>Request is <a
href="https://github.com/theekrystallee/hedera-style-guide/blob/sdk-v1/
deprecated/hedera-api/consensus-service/broken-reference/
README.md">ConsensusSubmitMessageTransactionBody</a></p> |

```

consensuscreatetopic.md:

ConsensusCreateTopic

ConsensusCreateTopicTransactionBody

Field	Type	Description
-----	-----	-----

```

-----
-----
-----
-----
| memo | string | Short publicly
visible memo about the topic. No guarantee of uniqueness.
|
| adminKey | Key | <p>Access control for
updateTopic/deleteTopic.<br>Anyone can increase the topic's expirationTime via
ConsensusService.updateTopic(), regardless of the adminKey.<br>If no adminKey is
specified, updateTopic may only be used to extend the topic's expirationTime,
and deleteTopic<br>is disallowed.</p>
|
| submitKey | Key | <p>Access control for
submitMessage.<br>If unspecified, no access control is performed on
ConsensusService.submitMessage (all submissions are allowed).</p>
|
| autoRenewPeriod | Duration | <p>The initial lifetime of the topic and the
amount of time to attempt to extend the topic's lifetime by<br>automatically at
the topic's expirationTime, if the autoRenewAccount is configured (once
autoRenew functionality<br>is supported by HAPI).<br>Limited to
MINAUTORENEWPERIOD and MAXAUTORENEWPERIOD value by server-side
configuration.<br>Required.</p>
|
| autoRenewAccount | AccountID | <p>Optional account to be used at the topic's
expirationTime to extend the life of the topic (once autoRenew<br>functionality
is supported by HAPI).<br>The topic lifetime will be extended up to a maximum of
the autoRenewPeriod or however long the topic<br>can be extended using all funds
on the account (whichever is the smaller duration/amount and if any
extension<br>is possible with the account's funds).<br>If specified, there must
be an adminKey and the autoRenewAccount must sign this transaction.</p> |

```

consensusdeletetopic.md:

ConsensusDeleteTopic

ConsensusDeleteTopicTransactionBody

Field	Type	Description
topicID	TopicID	Topic identifier.

consensusgettopicinfo.md:

ConsensusGetTopicInfo

ConsensusGetTopicInfoQuery

Field	Type	Description

```

-----
-----
| header | QueryHeader | Standard info sent from client to node, including the
signed payment, and what kind of response is requested (cost, state proof, both,
or neither). |
| topicID | TopicID | The Topic for which information is being
requested
|

```

ConsensusGetTopicInfoResponse

Retrieve the parameters of and state of a consensus topic.

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither.
topicID	TopicID	Topic identifier.
topicInfo	ConsensusTopicInfo	Current state of the topic

\\

consensussubmitmessage.md:

ConsensusSubmitMessage


ConsensusMessageChunkInfo

Field	Type	Description
initialTransactionID	TransactionID	TransactionID of the first chunk, gets copied to every subsequent chunk in a fragmented message.
total	int32	The total number of chunks in the message.
number	int32	The sequence number (from 1 to total) of the current chunk in the message.

ConsensusSubmitMessageTransactionBody

Field	Type	Description
topicID	TopicID	Topic to submit message to.
message	bytes	Message to be submitted. Max size of a message is 1024 bytes (1 kb).
chunkInfo	ConsensusMessageChunkInfo	Optional information of the current chunk in a fragmented message.

{% hint style="info" %}

 NOTE: Max size of a transaction (including messages and signatures) is 6kb.

{% endhint %}

consensustopicinfo.md:

ConsensusTopicInfo

Field	Type	Description
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
-		
memo		Short publicly visible memo about the topic. No guarantee of uniqueness.
runningHash		<p>When a topic is created, its running hash is initialized to 48 bytes of binary zeros. For each submitted message, the topic's running hash is then updated to the output of a particular SHA-384 digest whose input data include the previous running hash. See the TransactionReceipt.proto documentation for an exact description of the data included in the SHA-384 digest used for the update.</p>
sequenceNumber		Sequence number (starting at 1 for the first submitMessage) of messages on the topic.
expirationTime	Timestamp	Effective consensus timestamp at (and after) which submitMessage calls will no longer succeed on the topic and the topic will expire and after AUTORENEW\GRACE\PERIOD be automatically deleted.
adminKey	Key	Access control for update/delete of the topic. Null if there is no key.
submitKey	Key	Access control for ConsensusService.submitMessage. Null if there is no key.
autoRenewPeriod	Duration	The duration in which to renew the topic
autoRenewAccount	AccountID	Null if there is no autoRenewAccount.
ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs.

consensusupdatetopic.md:

ConsensusUpdateTopic

All fields left null will not be updated.

ConsensusUpdateTopicTransactionBody

Field	Type	Description
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
topicID	TopicID	UNDOCUMENTED

memo	google.protobuf.StringValue	Short publicly visible memo about the topic. No guarantee of uniqueness. Null for "do not update".
expirationTime	Timestamp	Effective consensus timestamp at (and after) which all consensus transactions and queries will fail. The expirationTime may be no longer than MAXAUTORENEWPERIOD (8000001 seconds) from the consensus timestamp of this transaction. On topics with no adminKey, extending the expirationTime is the only updateTopic option allowed on the topic. If unspecified, no change.
adminKey	Key	Access control for update/delete of the topic. If unspecified, no change. If empty keyList - the adminKey is cleared.
submitKey	Key	Access control for ConsensusService.submitMessage. If unspecified, no change. If empty keyList - the submitKey is cleared.
autoRenewPeriod	Duration	The amount of time to extend the topic's lifetime automatically at expirationTime if the autoRenewAccount is configured and has funds (once autoRenew functionality is supported by HAPI). Limited to between MINAUTORENEWPERIOD (6999999 seconds) and MAXAUTORENEWPERIOD (8000001 seconds) by servers-side configuration (which may change). If unspecified, no change.
autoRenewAccount	AccountID	Optional account to be used at the topic's expirationTime to extend the life of the topic. Once autoRenew functionality is supported by HAPI, the topic lifetime will be extended up to a maximum of the autoRenewPeriod or however long the topic can be extended using all funds on the account (whichever is the smaller duration/amount). If specified as the default value (0.0.0), the autoRenewAccount will be removed. If unspecified, no change.
memo	string	The memo associated with the topic (UTF-8 encoding max 100 bytes)

README (1).md:

Consensus Service

README.md:

Consensus Service

cryptapproveallowance.md:

CryptApproveAllowance

CryptApproveAllowanceTransactionBody

Creates one or more hbar/token approved allowances relative to the payer account of this transaction. Each allowance grants a spender the right to transfer a pre-determined amount of the payer's hbar/token to any other account of the spender's choice. (So if account 0.0.X pays for this transaction and owner is not specified in the allowance, then at consensus each spender account will have new allowances to spend hbar or tokens from 0.0.X).

Field	Type
-------	------

Description	
cryptoAllowance	repeated CryptoAllowance List of hbar allowances approved by the account owner
nftAllowance	repeated NftAllowance List of non-fungible token allowances approved by the account owner
tokenAllowance	repeated TokenAllowance List of fungible token allowances approved by the account owner.

cryptocreate.md:

CryptoCreate

CryptoCreateTransactionBody

Create a new account. After the account is created, the AccountID for it is in the receipt, or can be retrieved with a GetByKey query, or by asking for a Record of the transaction to be created, and retrieving that. The account can then automatically generate records for large transfers into it or out of it, which each last for 25 hours. Records are generated for any transfer that exceeds the thresholds given here. This account is charged cryptocurrency for each record generated, so the thresholds are useful for limiting Record generation to happen only for large transactions. The Key field is the key used to sign transactions for this account. If the account has receiverSigRequired set to true, then all cryptocurrency transfers must be signed by this account's key, both for transfers in and out. If it is false, then only transfers out have to be signed by it. When the account is created, the payer account is charged enough hbars so that the new account will not expire for the next autoRenewPeriod seconds. When it reaches the expiration time, the new account will then be automatically charged to renew for another autoRenewPeriod seconds. If it does not have enough hbars to renew for that long, then the remaining hbars are used to extend its expiration as long as possible. If it is has a zero balance when it expires, then it is deleted. This transaction must be signed by the payer account. If receiverSigRequired is false, then the transaction does not have to be signed by the keys in the keys field. If it is true, then it must be signed by them, in addition to the keys of the payer account.

Field	Type
Description	
key	Key The key that must sign each transfer out of the account. If receiverSigRequired is true, then it must also sign any transfer into the account.
initialBalance	uint64 The initial number of tinybars to put into the account
proxyAccountID	AccountID ID of the account to which this account is proxy staked. If proxyAccountID is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking, or if it is not currently running a node, then it will behave as if proxyAccountID was null.

sendRecordThreshold	uint64	
\[Deprecated v0.8.0] The threshold amount (in tinybars) for which an account record is created for any send/withdraw transaction		
receiveRecordThreshold	uint64	
\[Deprecated v0.8.0] The threshold amount (in tinybars) for which an account record is created for any receive/deposit transaction		
receiverSigRequired	bool	
If true, this account's key must sign any transaction depositing into this account (in addition to all withdrawals)		
autoRenewPeriod	Duration	The account is charged to extend its expiration date every this many seconds. If it doesn't have enough balance, it extends as long as possible. If it is empty when it expires, then it is deleted.
shardID	ShardID	The shard in which this account is created
realmID	RealmID	The realm in which this account is created (leave this null to create a new realm)
newRealmAdminKey	Key	If realmID is null, then this the admin key for the new realm that will be created
memo	string	
The memo associated with the account (UTF-8 encoding max 100 bytes)		
maxautomatictokenassociations	int32	The maximum number of tokens that an Account can be implicitly associated with. Defaults to 0 and up to a maximum value of 1000.

cryptodelete.md:

CryptoDelete

CryptoDeleteTransactionBody

Mark an account as deleted, moving all its current HBAR to another account. It will remain in the ledger, marked as deleted, until it expires. Transfers into it a deleted account fail. But a deleted account can still have its expiration extended in the normal way.

Field	Type	Description
transferAccountID	AccountID	The account ID which will receive all remaining hbars
deleteAccountID	AccountID	The account ID which should be deleted

cryptodeleteallowance.md:

CryptoDeleteAllowance

Deletes one or more non-fungible approved allowances from an owner's account. This operation will remove the allowances granted to one or more specific non-fungible token serial numbers. Each owner account listed as wiping an allowance must sign the transaction. Hbar and fungible token allowances can be removed by

setting the amount to zero in CryptoApproveAllowance.

Field	Type	Description
-----	-----	-----
nftAllowances	repeated NFTRemoveAllowance	List of non-fungible token allowances to remove.

NFTRemoveAllowance

Nft allowances to be removed on an owner account.

Field	Type	Description
-----	-----	-----
tokenId	TokenID	The token that the allowance pertains to.
owner	AccountID	The account ID of the token owner (ie. the grantor of the allowance).
serialnumbers	repeated int64	The list of serial numbers to remove allowances from.

cryptogetaccountbalance.md:

CryptoGetAccountBalance

CryptoGetAccountBalanceQuery

Get the balance of a cryptocurrency account. This returns only the balance, so it is a smaller and faster reply than CryptoGetInfo, which returns the balance plus additional information.

Field	Type	Description
-----	-----	-----
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
one of:		
accountID	AccountID	The account ID for which information is requested
contractID	ContractID	The contract ID for which information is requested

CryptoGetAccountBalanceResponse

Response when the client sends the node CryptoGetAccountBalanceQuery

Field	Type	Description
-----	-----	-----
header	ResponseHeader	Standard response from node to client,

including the requested fields: cost, or state proof, or both, or neither |
| accountID | AccountID | The account ID that is being described
(this is useful with state proofs, for proving to a third party) |
| balance | uint64 | The
current balance, in tinybars
|
| tokenBalances | TokenBalance | The array of tokens that the account
possesses |

cryptogetaccountrecords.md:

CryptoGetAccountRecords

CryptoGetAccountRecordsQuery

Get all the records for an account for any transfers into it and out of it, that were above the threshold, during the last 25 hours.

Field	Type	Description
-------	------	-------------

header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
accountID	AccountID	The account ID for which the records should be retrieved

CryptoGetAccountRecordsResponse

Returns records of all transactions for which the given account was the effective payer in the last 3 minutes of consensus time and ledger.keepRecordsInState=true was true during handleTransaction.

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
accountID	AccountID	The account that this record is for
records	repeated TransactionRecord	List of records

cryptogetinfo.md:

CryptoGetInfo

CryptoGetInfoQuery

Get all the information about an account, including the balance. This does not get the list of account records.

Field	Type	Description
-------	------	-------------

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
accountID	AccountID	The account ID for which information is requested

CryptoGetInfoResponse

Response when the client sends the node CryptoGetInfoQuery

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither

AccountInfo

Response when the client sends the node CryptoGetInfoQuery

Field	Type	Description
accountID	AccountID	The account ID for which this information applies
contractAccountID	string	The Contract Account ID comprising of both the contract instance and the cryptocurrency account owned by the contract instance, in the format used by Solidity
deleted	bool	If true, then this account has been deleted, it will disappear when it expires, and all transactions for it will fail except the transaction to extend its expiration date
proxyAccountID	AccountID	The Account ID of the account to which this is proxy staked. If proxyAccountID is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking, or if it is not currently running a node, then it will behave as if proxyAccountID was null.
proxyReceived	int64	The total number of tinybars proxy staked to this account
key	Key	The key for the account, which must sign in order to transfer out, or to modify the account in any way other than extending its expiration date.

```

| balance | uint64
| The current balance of account in tinybars
|
| generateSendRecordThreshold | uint64
| \[Deprecated v0.8.0] The threshold amount (in tinybars) for which an account
record is created (and this account charged for them) for any send/withdraw
transaction.
|
| generateReceiveRecordThreshold | uint64
| \[Deprecated v0.8.0] The threshold amount (in tinybars) for which an account
record is created (and this account charged for them) for any transaction above
this amount.
|
| receiverSigRequired |
| If true, no transaction can transfer to this account unless signed by this
account's key
|
| expirationTime | Timestamp | The TimeStamp
time at which this account is set to expire
|
| autoRenewPeriod | Duration | The duration for
expiration time will extend every this many seconds. If there are insufficient
funds, then it extends as long as possible. If it is empty when it expires, then
it is deleted.
|
| liveHashes | Claim
| All of the livehashes attached to the account (each of which is a hash along
with the keys that authorized it and can delete it )
|
| tokenRelationships | TokenRelationship | All tokens related to
this account
|
| memo | string
| The memo associated with the account
|
| ownedNfts | int64
| The number of NFTs owned by this account
|
| maxautomatictokenassociations | int32
| The maximum number of tokens that an Account can be implicitly associated with
|
| alias | bytes
| The alias of this account
|
| ledgerid | bytes
| The ledger ID the response was returned from; please see HIP-198 for the
network-specific IDs.
|

```

####

cryptogetstakers.md:

CryptoGetStakers

AllProxyStakers

all of the accounts proxy staking to a given account, and the amounts proxy
staked

Field	Type	Description

-----	-----	
accountID	AccountID	The Account ID that is being proxy staked to
proxyStaker	ProxyStaker	Each of the proxy staking accounts, and the amount they are proxy staking

CryptoGetStakersQuery

Get all the accounts that are proxy staking to this account. For each of them, give the amount currently staked. This is not yet implemented, but will be in a future version of the API.

Field	Type	Description
-----	-----	

header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
accountID	AccountID	The Account ID for which the records should be retrieved

CryptoGetStakersResponse

Response when the client sends the node CryptoGetStakersQuery

Field	Type	Description
-----	-----	
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
stakers	AllProxyStakers	List of accounts proxy staking to this account, and the amount each is currently proxy staking

ProxyStaker

information about a single account that is proxy staking

Field	Type	Description
-----	-----	
accountID	AccountID	The Account ID that is proxy staking
amount	int64	The number of hbars that are currently proxy staked

cryptoservice.md:

CryptoService

RPC	Request	Comments
Response		
-----	-----	

createAccount	Transaction	TransactionResponse	Creates a new account by submitting the transaction. The grpc server returns the TransactionResponse
updateAccount	Transaction	TransactionResponse	Updates an account by submitting the transaction. The grpc server returns the TransactionResponse
cryptoTransfer	Transaction	TransactionResponse	Initiates a transfer by submitting the transaction. The grpc server returns the TransactionResponse
cryptoDelete	Transaction	TransactionResponse	Deletes an account by submitting the transaction. The grpc server returns the TransactionResponse
addLiveHash	Transaction	TransactionResponse	(NOT CURRENTLY SUPPORTED) Adds a livehash
deleteLiveHash	Transaction	TransactionResponse	(NOT CURRENTLY SUPPORTED) Deletes a livehash
getLiveHash	Query	Response	(NOT CURRENTLY SUPPORTED) Retrieves a livehash for an account
getAccountRecords	Query	Response	Retrieves the record(fetch by AccountID ID) for an account by submitting the query.
cryptoGetBalance	Query	Response	Retrieves the balance for an account by submitting the query.
getAccountInfo	Query	Response	Retrieves the account information for an account by submitting the query.
getTransactionReceipts	Query	Response	Retrieves the transaction receipts for an account by TxId which last for 180sec only for no fee.
getFastTransactionRecord	Query	Response	Retrieves the transaction record by TxID which last for 180sec only for no fee.
getTxRecordByTxID	Query	Response	Retrieves the transactions record(fetch by Transaction ID) for an account by submitting the query.
getStakersByAccountID	Query	Response	Retrieves the stakers for a node by account ID by submitting the query.
approveAllowances	Query	Response	Adds one or more approved allowances for spenders to transfer the paying account's hbar or tokens
deleteAllowance	Query	Response	Deletes the approved NFT allowances on an owner account

cryptotransfer.md:

CryptoTransfer

Transfer cryptocurrency from some accounts to other accounts. The accounts list can contain up to 10 accounts. The amounts list must be the same length as the accounts list. Each negative amount is withdrawn from the corresponding account (a sender), and each positive one is added to the corresponding account (a receiver). The amounts list must sum to zero. Each amount is a number of tinyBars (there are 100,000,000 tinyBars in one Hbar). If any sender account fails to have sufficient hbars to do the withdrawal, then the entire transaction fails, and none of those transfers occur, though the transaction fee is still charged. This transaction must be signed by the keys for all the sending accounts, and for any receiving accounts that have receiverSigRequired == true.

The signatures are in the same order as the accounts, skipping those accounts that don't need a signature.

CryptoTransferTransactionBody

```

| Field | Type
| Description
| -----
|
|-----
|
|-----
| transfers | TransferList | Accounts and amounts to
transfer
|
| tokenTransfers | repeated TokenTransferList | The desired token unit balance
adjustments; if any custom fees are assessed, the ledger will try to deduct them
from the payer of this CryptoTransfer, resolving the transaction to
INSUFFICIENTPAYERBALANCEFORCUSTOMFEE if this is not possible |

```

```
# cryptoupdate.md:
```

CryptoUpdate

CryptoUpdateTransactionBody

Change properties for the given account. Any null field is ignored (left unchanged). This transaction must be signed by the existing key for this account. If the transaction is changing the key field, then the transaction must be signed by both the old key (from before the change) and the new key. The old key must sign for security. The new key must sign as a safeguard to avoid accidentally changing to an invalid key, and then having no way to recover. When extending the expiration date, the cost is affected by the size of the list of attached claims, and of the keys associated with the claims and the account.

Field	Type
Description	

accountIDToUpdate	AccountID
updated in this transaction	The account ID which is being updated
key	Key
	The new key
proxyAccountID	AccountID
this account is proxy staked. If proxyAccountID is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking , or if it is not currently running a node, then it will behave as if proxyAccountID was null.	
proxyFraction	
\[Deprecated]. payments earned from proxy staking are shared between the node and this account, with proxyFraction / 10000 going to this account	

sendRecordThresholdField	one of:	\
[Deprecated v0.8.0]		
	sendRecordThreshold	
	sendRecordThresholdWrapper	
google.protobuf.UInt64Value		
receiveRecordThresholdField	one of:	\
[Deprecated v0.8.0]		
	receiveRecordThreshold	
	receiveRecordThresholdWrapper	
google.protobuf.UInt64Value		
autoRenewPeriod	Duration	The duration in which it will
automatically extend the expiration period. If it doesn't have enough balance,		
it extends as long as possible. If it is empty when it expires, then it is		
deleted.		
expirationTime	Timestamp	The new expiration time to
extend to (ignored if equal to or before the current one)		
receiverSigRequiredField	one of:	
	receiverSigRequired	
\[Deprecated] Do NOT use this field to set a false value because the server		
cannot distinguish from the default value. Use receiverSigRequiredWrapper field		
for this purpose.		
	receiverSigRequiredWrapper	
google.protobuf.BoolValue		
memo	string	
The memo associated with the account (UTF-8 encoding max 100 bytes)		
maxautomatictokenassociations	google.protobuf.Int32Value	
The maximum number of tokens that an Account can be implicitly associated with.		
Up to a 1000 including implicit and explicit associations.		

README.md:

Cryptocurrency Accounts

README.md:

File Service

duration.md:

Duration

A length of time in seconds.

Field	Description	
-------	-------------	--

	-----		-----	
	seconds		The number of seconds	

exchangerate.md:

ExchangeRate

ExchangeRate

An exchange rate between hbar and cents (USD) and the time at which the exchange rate will expire, and be superseded by a new exchange rate.

	Field		Type		
	Description				
	-----		-----		
	-----		-----		
	hbarEquiv				
	Denominator in calculation of exchange rate between hbar and cents				
	centEquiv				Numerator
	in calculation of exchange rate between hbar and cents				
	expirationTime		TimestampSeconds		Expiration time in seconds for this exchange rate

ExchangeRateSet

Two sets of exchange rate

	Field		Type		Description
	-----		-----		
	-----		-----		
	currentRate		ExchangeRate		Current exchange rate
	nextRate		ExchangeRate		Next exchange rate which will take effect when current rate expires

freeze.md:

Freeze

FreezeTransactionBody

Set the freezing period in which the platform will stop creating events and accepting transactions. This is used before safely shut down the platform for maintenance.

	Field		Description

	startHour[deprecated=true]		The start hour (in UTC time), a value between 0 and 23
	startMin[deprecated=true]		The start minute (in UTC time), a value between 0 and 59
	endHour[deprecated=true]		The end hour (in UTC time), a value between 0 and 23
	endMin[deprecated=true]		The end minute (in UTC time), a value between 0 and 59
	updatefile		If set, the file whose contents should be used for a network software update during the maintenance window.
	filehash		If set, the expected hash of the contents of the

update file (used to verify the update).	
starttime	The consensus time at which the maintenance window should begin.
freezetype	The type of network freeze or upgrade operation to perform.

FreezeService.proto

FreezeService

RPC	Request	Response	Comments
freeze	Transaction	TransactionResponse	Freezes the nodes by submitting the transaction. The grpc server returns the TransactionResponse

freezetype.md:

FreezeType

The type of network freeze or upgrade operation to be performed. This type dictates which fields are required.

Enum	Description
UNKNOWNFREEZETYPE	An (invalid) default value for this enum, to ensure the client explicitly sets the intended type of freeze transaction.
FREEZEONLY	Freezes the network at the specified time. The start\time field must be provided and must reference a future time. Any values specified for the update\file and file\hash fields will be ignored. This transaction does not perform any network changes or upgrades and requires manual intervention to restart the network.
PREPAREUPGRADE	A non-freezing operation that initiates network wide preparation in advance of a scheduled freeze upgrade. The update\file and file\hash fields must be provided and valid. The start\time field may be omitted and any value present will be ignored.
FREEZEUPGRADE	Freezes the network at the specified time and performs the previously prepared automatic upgrade across the entire network.
FREEZEABORT	Aborts a pending network freeze operation.
TELEMETRYUPGRADE	Performs an immediate upgrade on auxiliary services and containers providing telemetry/metrics. Does not impact network operations.

getbykey.md:

GetByKey

EntityID

the ID for a single entity (account, claim, file, or smart contract instance)

Field	Type	Description
-----	-----	-----
accountID	AccountID	The Account ID for the cryptocurrency account
liveHash	LiveHash	A uniquely identifying livehash of an account
fileID	FileID	The file ID of the file
contractID	ContractID	The smart contract ID that identifies instance

GetByKeyQuery

Get all accounts, claims, files, and smart contract instances whose associated keys include the given Key. The given Key must not be a contractID or a ThresholdKey. This is not yet implemented in the API, but will be in the future.

Field	Type	Description
-----	-----	-----
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
key	Key	The key to search for. It must not contain a contractID nor a ThresholdSignature.

GetByKeyResponse

Response when the client sends the node GetByKeyQuery

Field	Type	Description
-----	-----	-----
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
entities	EntityID	The list of entities that include this public key in their associated Key list

getbysolidityid.md:

GetBySolidityID

GetBySolidityIDQuery

Get the IDs in the format used by transactions, given the ID in the format used by Solidity. If the Solidity ID is for a smart contract instance, then both the ContractID and associated AccountID will be returned.

Field	Type	Description
-----	-----	-----
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).

solidityID string	The ID in the format used by Solidity
---------------------	---------------------------------------

GetBySolidityIDResponse

Response when the client sends the node GetBySolidityIDQuery

Field	Type	Description
-----	-----	
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
accountID	AccountID	The Account ID for the cryptocurrency account
fileID	FileID	The file Id for the file
contractID	ContractID	A smart contract ID for the instance (if this is included, then the associated accountID will also be included)

networkgetversioninfo.md:

NetworkGetVersionInfo

Get the deployed versions of Hedera Services and the HAPI proto in semantic version format

NetworkGetVersionInfoQuery

Field	Type	Description
-----	-----	
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).

NetworkGetVersionInfoResponse

Response when the client sends the node NetworkGetVersionInfoQuery

Field	Type	Description
-----	-----	
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
hapiProtoVersion	SemanticVersion	The Hedera API (HAPI) protobuf version recognized by the responding node
hederaServicesVersion	SemanticVersion	The version of the Hedera Services software deployed on the responding node

networkservice.md:

NetworkService

The requests and responses for different network services.

NetworkService

RPC	Request	Response	Comments
-----	-----	-----	-----
getVersionInfo	Query	Response	Retrieves the active versions of Hedera Services and HAPI proto

query.md:

Query

Query

A single query, which is sent from the client to the node. This includes all possible queries. Each Query should not have more than 50 levels.

```
<table><thead><tr><th width="282">Field</th><th width="259.3333333333333">Type</th><th>Description</th></tr></thead><tbody><tr><td><code>getByKey</code></td><td><a href="getbykey.md">GetByKeyQuery</a></td><td>Get all entities associated with a given key</td></tr><tr><td><code>getBySolidityID</code></td><td><a href="getbysolidityid.md">GetBySolidityIDQuery</a></td><td>Get the IDs in the format used in transactions, given the format used in Solidity</td></tr><tr><td><code>contractCallLocal</code></td><td><a href="..readme-1-1/contractcalllocal.md">ContractCallLocalQuery</a></td><td>Call a function of a smart contract instance</td></tr><tr><td><code>contractGetInfo</code></td><td><a href="..readme-1-1/contractgetinfo.md">ContractGetInfoQuery</a></td><td>Get information about a smart contract instance</td></tr><tr><td><code>contractGetBytecode</code></td><td><a href="..readme-1-1/contractgetbytecode.md">ContractGetBytecodeQuery</a></td><td>Get bytecode used by a smart contract instance</td></tr><tr><td><code>ContractGetRecords</code></td><td><a href="..readme-1-1/contractgetrecords.md">ContractGetRecordsQuery</a></td><td>Get Records of the contract instance</td></tr><tr><td><code>cryptoGetAccountBalance</code></td><td><a href="..cryptocurrency-accounts/cryptogetaccountbalance.md">CryptoGetAccountBalanceQuery</a></td><td>Get the current balance in a cryptocurrency account</td></tr><tr><td><code>cryptoGetAccountRecords</code></td><td><a href="..cryptocurrency-accounts/cryptogetaccountrecords.md">CryptoGetAccountRecordsQuery</a></td><td>Get all the records that currently exist for transactions involving an account</td></tr><tr><td><code>cryptoGetInfo</code></td><td><a href="..cryptocurrency-accounts/cryptogetinfo.md">CryptoGetInfoQuery</a></td><td>Get all information about an account</td></tr><tr><td><code>cryptoGetLiveHash</code></td><td><a href="..cryptocurrency-accounts/cryptogetinfo.md">CryptoGetLiveHashQuery</a></td><td>Get a single livehash from a single account (or null if it doesn't exist)</td></tr><tr><td><code>cryptoGetProxyStakers</code></td><td><a href="..cryptocurrency-accounts/cryptogetstakers.md">CryptoGetStakersQuery</a></td><td>Get all the accounts that proxy stake to a given account, and how much they proxy stake (not yet implemented in the current API)</td></tr><tr><td><code>fileGetContents</code></td><td><a href="..readme-1/filegetcontents.md">FileGetContentsQuery</a></td><td>Get the contents of a file (the bytes stored in it)</td></tr><tr><td><code>fileGetInfo</code></td><td><a href="..readme-1/filegetinfo.md">FileGetInfoQuery</a></td><td>Get information about a file, such as its expiration
```



```

date</td></tr><tr><td><code>transactionGetReceipt</code></td><td><a
href="transactiongetreceipt.md">TransactionGetReceiptQuery</a></td><td>Get a
receipt for a transaction (lasts 180
seconds)</td></tr><tr><td><code>transactionGetRecord</code></td><td><a
href="transactiongetrecord.md">TransactionGetRecordQuery</a></td><td>Get a
record for a transaction (lasts 1
hour)</td></tr><tr><td><code>transactionGetFastRecord</code></td><td><a
href="transactiongetfastrecord.md">TransactionGetFastRecordQuery</a></td><td>Get
a record for a transaction (lasts 180
seconds)</td></tr><tr><td><code>consensusGetTopicInfo</code></td><td><a
href="..../consensus/consensusgettopicinfo.md">ConsensusGetTopicInfoQuery</a></
td><td>Get the parameters of and state of a consensus
topic.</td></tr><tr><td><code>networkGetVersionInfo</code></td><td><a
href="networkgetversioninfo.md">NetworkGetVersionInfoQuery</a></td><td>Get the
version of the network</td></tr><tr><td><code>tokenGetInfo</code></td><td><a
href="..../token-service/tokengetinfo.md#tokengetinfoquery">TokenGetInfoQuery</
a></td><td>Get all information about a
token</td></tr><tr><td><code>scheduleGetInfo</code></td><td><a
href="..../schedule-service/schedulegetinfo.md#schedulegetinfoquery">ScheduleGetIn
foQuery</a></td><td>Get all information about a schedule
entity</td></tr><tr><td><code>tokenGetAccountNftInfo</code></td><td><a
href="..../token-service/tokengetaccountnftinfo.md">TokenGetAccountNftInfoQuery</
a></td><td>Get a list of NFTs associated with the
account</td></tr><tr><td><code>tokenGetNftInfo</code></td><td><a href="..../token-
service/tokengetnftinfo.md#tokengetnftinfoquery">TokenGetNftInfoQuery</a></
td><td>Get all information about a
NFT</td></tr><tr><td><code>tokenGetNftInfos</code></td><td><a href="..../token-
service/tokengetnftinfo.md#tokengetnftinfoquery">TokenGetNftInfosQuery</a></
td><td>Get a list of NFTs for the token</td></tr></tbody></table>

```

queryheader.md:

QueryHeader

Each query from the client to the node will contain the QueryHeader, which gives the requested response type, and includes a payment for the response. It will sometimes leave payment blank: it is blank for TransactionGetReceiptQuery. It can also be left blank when the responseType is costAnswer or costAnswerStateProof. But it needs to be filled in for all other cases. The idea is that an answer that is only a few bytes (or that was paid for earlier) can be given for free. But if the answer is something that requires many bytes or much computation (like a state proof), then it should be paid for.

Field	Type	Description
-----	-----	-----
payment	Transaction	A signed CryptoTransferTransaction to pay the node a fee for handling this query
responseType	ResponseType	The requested response, asking for cost, state proof, both, or neither

ResponseType

The client uses the ResponseType to request that the node send it just the answer, or both the answer and a state proof. It can also ask for just the cost for getting the answer or both. If the payment in the query fails the precheck, then the response may have some fields blank. The state proof is only available for some types of information. It is available for a Record, but not a receipt. It is available for the information in each kind of \GetInfo request.

Enum Name	Description
-----------	-------------

Field	Type
ANSWERONLY	Response returns answer
ANSWERSTATEPROOF	Response returns both answer and state proof (not yet supported)
COSTANSWER	Response returns the cost of answer
COSTANSWERSTATEPROOF	Response returns the total cost of answer and state proof (not yet supported)

README.md:

Miscellaneous

response.md:

Response

A single response, which is returned from the node to the client, after the client sent the node a query. This includes all responses.

Field	Type
Description	

getByKey	GetByKeyResponse
Get all entities associated with a given key	
getBySolidityID	GetBySolidityIDResponse
Get the IDs in the format used in transactions, given the format used in Solidity	
contractCallLocal	ContractCallLocalResponse
Response to call a function of a smart contract instance	
contractGetBytecodeResponse	ContractGetBytecodeResponse
Get the bytecode for a smart contract instance	
contractGetInfo	ContractGetInfoResponse
Get information about a smart contract instance	
contractGetRecordsResponse	ContractGetRecordsResponse
Get all existing records for a smart contract instance	
cryptoGetAccountBalance	CryptoGetAccountBalanceResponse
Get the current balance in a cryptocurrency account	
cryptoGetAccountRecords	CryptoGetAccountRecordsResponse
Get all the records that currently exist for transactions involving an account	
cryptoGetInfo	CryptoGetInfoResponse
Get all information about an account	
cryptoGetLiveHash	CryptoGetLiveHashResponse
Get a single claim from a single account (or null if it doesn't exist)	
cryptoGetProxyStakers	CryptoGetStakersResponse

	Get all the accounts that proxy stake to a given account, and how much they proxy stake	
	fileGetContents	FileGetContentsResponse
	Get the contents of a file (the bytes stored in it)	
	fileGetInfo	FileGetInfoResponse
	Get information about a file, such as its expiration date	
	transactionGetReceipt	TransactionGetReceiptResponse
	Get a receipt for a transaction (lasts 180 seconds)	
	transactionGetRecord	TransactionGetRecordResponse
	Get a record for a transaction (lasts 1 hour)	
	transactionGetFastRecord	TransactionGetFastRecordResponse
	Get a record for a transaction (lasts 180 seconds)	
	consensusGetTopicInfo	ConsensusGetTopicInfoResponse
	Parameters of and state of a consensus topic.	
	networkGetVersionInfo	NetworkGetVersionInfoResponse
	Semantic versions of Hedera Services and HAPI proto	
	tokenGetInfo	TokenGetInfoResponse
	Get all information about a token	
	scheduleGetInfo	ScheduleGetInfoResponse
	Get all information about a schedule entity	
	tokenGetAccountNftInfo	TokenGetAccountNftInfoResponse
	A list of the NFTs associated with the account	
	tokenGetNftInfo	TokenGetNftInfoResponse
	All information about an NFT	
	tokenGetNftInfos	TokenGetNftInfosResponse
	A list of the NFTs for the token	

responsecode.md:

ResponseCode

<details>

<summary>OK</summary>

The transaction passed the precheck validations.

</details>

<details>

<summary>INVALIDTRANSACTION</summary>

For any error not handled by specific error codes listed below.

</details>

<details>

<summary>PAYERACCOUNTNOTFOUND</summary>

Payer account does not exist.

</details>

<details>

<summary>INVALIDNODEACCOUNT</summary>

Node Account provided does not match the node account of the node the transaction was submitted to.

</details>

<details>

<summary>TRANSACTIONEXPIRED</summary>

Pre-Check error when TransactionValidStart + transactionValidDuration is less than current consensus time.

</details>

<details>

<summary>INVALIDTRANSACTIONSTART</summary>

Transaction start time is greater than current consensus time

</details>

<details>

<summary>INVALIDTRANSACTIONDURATION</summary>

Invalid transaction duration is a positive non-zero number that does not exceed 120 seconds

</details>

<details>

<summary>INVALIDSIGNATURE</summary>

The transaction signature is not valid

</details>

<details>

<summary>MEMOTOOLONG</summary>

Transaction memo size exceeded 100 bytes

</details>

<details>

<summary>INSUFFICIENTTXFEE</summary>

The fee provided in the transaction is insufficient for this type of transaction

</details>

<details>

<summary>INSUFFICIENTPAYERBALANCE</summary>

The payer account has insufficient cryptocurrency to pay the transaction fee

</details>

<details>

<summary>DUPLICATETRANSACTION</summary>

This transaction ID is a duplicate of one that was submitted to this node or reached consensus in the last 180 seconds (receipt period)

</details>

<details>

<summary>BUSY</summary>

If API is throttled out

</details>

<details>

<summary>NOTSUPPORTED</summary>

The API is not currently supported

</details>

<details>

<summary>INVALIDFILEID</summary>

The file id is invalid or does not exist

</details>

<details>

<summary>INVALIDACCOUNTID</summary>

The account id is invalid or does not exist

</details>

<details>

<summary>INVALIDCONTRACTID</summary>

The contract id is invalid or does not exist

</details>

<details>

<summary>INVALIDTRANSACTIONID</summary>

Transaction id is not valid

</details>

<details>

<summary>RECEIPTNOTFOUND</summary>

Receipt for given transaction id does not exist

</details>

<details>

<summary>RECORDNOTFOUND</summary>

Record for given transaction id does not exist

</details>

<details>

<summary>INVALIDSOLIDITYID</summary>

The solidity id is invalid or entity with this solidity id does not exist

</details>

<details>

<summary>UNKNOWN</summary>

This node has submitted this transaction to the network. Status of the transaction is currently unknown.

</details>

<details>

<summary>SUCCESS</summary>

The transaction succeeded

</details>

<details>

<summary>FAILINVALID</summary>

There was a system error and the transaction failed because of invalid request parameters.

</details>

<details>

<summary>FAILFEE</summary>

There was a system error while performing fee calculation, reserved for future.

</details>

<details>

<summary>FAILBALANCE</summary>

There was a system error while performing balance checks, reserved for future.

</details>

<details>

<summary>KEYREQUIRED</summary>

Key not provided in the transaction body

</details>

<details>

<summary>BADENCODING</summary>

Unsupported algorithm/encoding used for keys in the transaction

</details>

<details>

<summary>INSUFFICIENTACCOUNTBALANCE</summary>

When the account balance is not sufficient for the transfer

</details>

<details>

<summary>INVALIDSOLIDITYADDRESS</summary>

During an update transaction when the system is not able to find the Users Solidity address

</details>

<details>

<summary>INSUFFICIENTGAS</summary>

Not enough gas was supplied to execute transaction

</details>

<details>

<summary>CONTRACTSIZELIMITEXCEEDED</summary>

contract byte code size is over the limit

</details>

<details>

<summary>LOCALCALLMODIFICATIONEXCEPTION</summary>

local execution (query) is requested for a function which changes state

</details>

<details>

<summary>CONTRACTREVERTEXECUTED</summary>

Contract REVERT OPCODE executed

</details>

<details>

<summary>CONTRACTEXECUTIONEXCEPTION</summary>

For any contract execution-related error not handled by the specific error codes listed above.

</details>

<details>

<summary>INVALIDRECEIVINGNODEACCOUNT</summary>

In Query validation, an account with +ve(amount) value should be a Receiving node account, the receiver account should be only one account in the list.

</details>

<details>

<summary>MISSINGQUERYHEADER</summary>

The header is missing in the Query request.

</details>

<details>

<summary>ACCOUNTUPDATEFAILED</summary>

The update of the account failed.

</details>

<details>

<summary>INVALIDKEYENCODING</summary>

Provided key encoding was not supported by the system.

</details>

<details>

<summary>NULLSOLIDITYADDRESS</summary>

null solidity address

</details>

<details>

<summary>CONTRACTUPDATEFAILED</summary>

update of the contract failed

</details>

<details>

<summary>INVALIDQUERYHEADER</summary>

the query header is invalid

</details>

<details>

<summary>INVALIDFEESUBMITTED</summary>

Invalid fee submitted

</details>

<details>

<summary>INVALIDPAYERSIGNATURE</summary>

Payer signature is invalid

</details>

<details>

<summary>KEYNOTPROVIDED</summary>

The keys were not provided in the request.

</details>

<details>

<summary>INVALIDEXPIRATIONTIME</summary>

Expiration time provided in the transaction was invalid.

</details>

<details>

<summary>NOWACLKEY</summary>

WriteAccess Control Keys are not provided for the file

</details>

<details>

<summary>FILECONTENTEMPTY</summary>

The contents of file are provided as empty.

</details>

<details>

<summary>INVALIDACCOUNTAMOUNTS</summary>

The crypto transfer credit and debit do not sum equal to 0

</details>

<details>

<summary>EMPTYTRANSACTIONBODY</summary>

Transaction body provided is empty

</details>

responseheader.md:

ResponseHeader

ResponseHeader

Every query receives a response containing the QueryResponseHeader. Either or both of the cost and stateProof fields may be blank, if the responseType didn't ask for the cost or stateProof.

Field	Type	
Description		
----- -----		

nodeTransactionPrecheckCode ResponseCodeEnum Result of fee transaction precheck, saying it passed, or why it failed		
responseType ResponseType The requested response is repeated back here, for convenience		
cost uint64 The fee that would be charged to get the requested information (if a cost was requested). Note: This cost only includes the query fee and does not include the transfer fee(which is required to execute the transfer transaction to debit the payer account and credit the node account with query fee)		
stateProof bytes The state proof for this information (if a state proof was requested, and is available)		

systemdelete.md:

SystemDelete

SystemDeleteTransactionBody

Delete a file or smart contract - can only be done with a Hedera administrative multisignature. When it is deleted, it immediately disappears from the system as seen by the user, but is still stored internally until the expiration time, at which time it is truly and permanently deleted. Until that time, it can be undeleted by the Hedera administrative multisignature. When a smart contract is deleted, the cryptocurrency account within it continues to exist, and is not affected by the expiration time here.

Field	Type	Description
----- -----		

fileId FileID The file ID of the file to delete, in the format used in transactions		
contractId ContractID The contract ID instance to delete, in the format used in transactions		
expirationTime TimestampSeconds The timestamp in seconds at		

which the "deleted" file should truly be permanently deleted |

systemundelete.md:

SystemUndelete

SystemUndeleteTransactionBody

Undelete a file or smart contract that was deleted by SystemDelete; requires a Hedera administrative multisignature.

Field	Type	Description
-----	-----	
fileID	FileID	The file ID to undelete, in the format used in transactions
contractID	ContractID	The contract ID instance to undelete, in the format used in transactions

timestamp.md:

TimeStamp

TokenUnitBalance

Field	Description

tokenID	A unique token id
balance	Number of transferable units of the identified token. For token of type FUNGIBLECOMMON - balance in the smallest denomination. For token of type NONFUNGIBLEUNIQUE - the number of NFTs held by the account.

Timestamp

An exact date and time. This is the same data structure as the protobuf Timestamp.proto (see the comments in <https://github.com/google/protobuf/blob/master/src/google/protobuf/timestamp.proto>)

Field	Description
seconds	Number of complete seconds since the start of the epoch
nanos	Number of nanoseconds since the start of the last second

TimestampSeconds

An exact date and time, with a resolution of one second (no nanoseconds).

Field	Description
seconds	Number of complete seconds since the start of the epoch

transaction-contents.md:

TransactionContents

SignedTransaction

Field	Type	Description
-----	-----	-----
bodyBytes		TransactionBody serialized into bytes, which needs to be signed
sigMap	SignatureMap	The signatures on the body with the new format, to authorize the transaction

transaction.md:

Transaction

A single signed transaction, including all its signatures. The SignatureList will have a Signature for each Key in the transaction, either explicit or implicit, in the order that they appear in the transaction. For example, a CryptoTransfer will first have a Signature corresponding to the Key for the paying account, followed by a Signature corresponding to the Key for each account that is sending or receiving cryptocurrency in the transfer. Each Transaction should not have more than 50 levels.\n\nThe SignatureList field is deprecated and succeeded by SignatureMap.

Transaction

Field	Type	Description
-----	-----	-----
signedTransactionBytes	bytes	SignedTransaction serialized into bytes
bodyBytes	bytes	TransactionBody serialized into bytes, which needs to be signed deprecated = true
sigMap	SignatureMap	The signatures on the body with the new format, to authorize the transaction deprecated = true

transactionbody.md:

TransactionBody

A single transaction. All transaction types are possible here.

Field	Type	Description
-----	-----	-----
transactionID	TransactionID	The ID for this transaction, which includes the payer's account (the account paying the transaction fee). If two transactions have the same transactionID, they won't both have an effect
nodeAccountID	AccountID	The account of the node that submits the client's transaction to the network

transactionFee	uint64	The
maximum transaction fee the client is willing to pay		
transactionValidDuration	Duration	The
transaction is invalid if consensusTimestamp > transactionID.transactionValidStart + transactionValidDuration		
generateRecord	bool	Should a
record of this transaction be generated? (A receipt is always generated, but the record is optional)		
memo	string	Any notes
or descriptions that should be put into the record (max length 100)		
contractCall	ContractCallTransactionBody	Calls a
function of a contract instance		
contractCreateInstance	ContractCreateTransactionBody	Creates a
contract instance		
contractUpdateInstance	ContractUpdateTransactionBody	Updates a
contract		
contractDeleteInstance	ContractDeleteTransactionBody	Delete
contract and transfer remaining balance into specified account		
cryptoAddLiveHash	CryptoAddLiveHashTransactionBody	Attach a
new livehash to an account		
cryptoApproveAllowance	CryptoApproveAllowanceTransactionBody	Adds one
or more approved allowances for spenders to transfer the paying account's hbar or tokens.		
cryptoDeleteAllowance	CryptoDeleteAllowanceTransactionBody	Deletes
one or more approved hbar or token allowances from an owner's account		
cryptoCreateAccount	CryptoCreateTransactionBody	Create a
new cryptocurrency account		
cryptoDelete	CryptoDeleteTransactionBody	Delete a
cryptocurrency account (mark as deleted, and transfer hbars out)		
cryptoDeleteLiveHash	CryptoDeleteLiveHashTransactionBody	Remove a
livehash from an account		
cryptoTransfer	CryptoTransferTransactionBody	Transfer
amount between accounts		
cryptoUpdateAccount	CryptoUpdateTransactionBody	Modify
information such as the expiration date for an account		
fileAppend	FileAppendTransactionBody	Add bytes
to the end of the contents of a file		
fileCreate	FileCreateTransactionBody	Create a
new file		
fileDelete	FileDeleteTransactionBody	Delete a
file (remove contents and mark as deleted until it expires)		
fileUpdate	FileUpdateTransactionBody	Modify
information such as the expiration date for a file		
systemDelete	SystemDeleteTransactionBody	Hedera

administrative deletion of a file or smart contract

systemUndelete undelete an entity deleted by	SystemUndeleteTransactionBody SystemDelete	To
freeze nodes	FreezeTransactionBody	Freeze the
consensusCreateTopic topic	ConsensusCreateTopicTransactionBody	Creates a
consensusUpdateTopic topic	ConsensusUpdateTopicTransactionBody	Updates a
consensusDeleteTopic topic	ConsensusDeleteTopicTransactionBody	Deletes a
consensusSubmitMessage message to a topic	ConsensusSubmitMessageTransactionBody	Submits
uncheckedSubmit	UncheckedSubmitBody	
tokenCreation token instance	TokenCreateTransactionBody	Creates a
tokenTransfers tokens between accounts	TokenTransfersTransactionBody	Transfers
tokenFreeze account not to be able to transact with a token	TokenFreezeAccountTransactionBody	Freezes
tokenUnfreeze account for a token	TokenUnfreezeAccountTransactionBody	Unfreezes
tokenGrantKyc to an account for a token	TokenGrantKycTransactionBody	Grants KYC
tokenRevokeKyc KYC of an account for a token	TokenRevokeKycTransactionBody	Revokes
tokenDeletion token instance	TokenDeleteTransactionBody	Deletes a
tokenUpdate token instance	TokenUpdateTransactionBody	Updates a
tokenMint tokens to a token's treasury account	TokenMintTransactionBody	Mints new
tokenBurn tokens from a token's treasury account	TokenBurnTransactionBody	Burns
tokenWipe amount of tokens from an account	TokenWipeAccountTransactionBody	Wipes
tokenAssociate tokens to an account	TokenAssociateTransactionBody	Associate
tokenDissociate tokens from an account	TokenDissociateTransactionBody	Dissociate
tokenfeescheduleupdate token's custom fee schedule	TokenFeeScheduleUpdateTransactionBody	Updates a

tokenpause Token	TokenPauseTransactionBody	Pauses the
tokenunpause the Token	TokenUnpauseTransactionBody	Unpauses
scheduleCreate scheduled transaction instance	ScheduleCreateTransactionBody	Creates a
scheduleDelete scheduled transaction instance	ScheduleDeleteTransactionBody	Deletes a
scheduleSign scheduled transaction instance	ScheduleSignTransactionBody	Signs a
cryptoAdjustAllowance the approved allowance for a spender to transfer the paying account's hbar or tokens	CryptoAdjustAllowanceTransactionBody	Adjusts
cryptoApproveAllowance or more approved allowances for spenders to transfer the paying account's hbar or tokens	CryptoApproveAllowanceTransactionBody	Adds one

transactiongetfastrecord.md:

TransactionGetFastRecord

TransactionGetFastRecordQuery

Get the tx record of a transaction, given its transaction ID. Once a transaction reaches consensus, then information about whether it succeeded or failed will be available until the end of the receipt period. Before and after the receipt period, and for a transaction that was never submitted, the receipt is unknown. This query is free (the payment field is left empty).

Field	Type	
Description		
-----	-----	
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
transactionID	TransactionID	The ID of the transaction for which the record is requested.

TransactionGetFastRecordResponse

Response when the client sends the node TransactionGetFastRecordQuery. If it created a new entity (account, file, or smart contract instance) then one of the three ID fields will be filled in with the ID of the new entity. Sometimes a single transaction will create more than one new entity, such as when a new contract instance is created, and this also creates the new account that it owned by that instance.

Field	Type	Description
-----	-----	

header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
transactionRecord	TransactionRecord	The requested transaction records

transactiongetreceipt.md:

TransactionGetReceipt

TransactionGetReceiptQuery

Get the receipt of a transaction, given its transaction ID. Once a transaction reaches consensus, then information about whether it succeeded or failed will be available until the end of the receipt period. Before and after the receipt period, and for a transaction that was never submitted, the receipt is unknown. This query is free (the payment field is left empty). No State proof is available for this response

Field	Type	
Description		
-----	-----	
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
transactionID	TransactionID	The ID of the transaction for which the receipt is requested.
includeDuplicates	bool	Whether receipts of processing duplicate transactions should be returned along with the receipt of processing the first consensus transaction with the given id whose status was neither INVALID\NODE\ACCOUNT nor INVALID\PAYER\SIGNATURE; or, if no such receipt exists, the receipt of processing the first transaction to reach consensus with the given transaction id.
includechildreceipts	bool	Whether the response should include the receipts of any child transactions spawned by the top-level transaction with the given transactionID.

TransactionGetReceiptResponse

Response when the client sends the node TransactionGetReceiptQuery. If it created a new entity (account, file, or smart contract instance) then one of the three ID fields will be filled in with the ID of the new entity. Sometimes a single transaction will create more than one new entity, such as when a new contract instance is created, and this also creates the new account that it owned by that instance. No State proof is available for this response

Field	Type
Description	

```

----- |
| header | ResponseHeader | Standard
response from node to client, including the requested fields: cost, or state
proof, or both, or neither
|
| receipt | TransactionReceipt | The receipt,
indicating it reached consensus (and whether it succeeded or failed) or is
currently unknown (because it hasn't reached consensus yet, or the transaction
has expired already), and including the ID of any new account/file/instance
created by that transaction. |
| duplicateTransactionReceipts | TransactionReceipt | The receipts of
processing all transactions with the given id, in consensus time order.
|
| childtransactionreceipts | repeated TransactionReceipt | The receipts (if
any) of all child transactions spawned by the transaction with the given top-
level id, in consensus order. Always empty if the top-level status is UNKNOWN.
|

```

transactiongetrecord.md:

TransactionGetRecord

TransactionGetRecordQuery

Get the record for a transaction. If the transaction requested a record, then the record lasts for one hour, and a state proof is available for it. If the transaction created an account, file, or smart contract instance, then the record will contain the ID for what it created. If the transaction called a smart contract function, then the record contains the result of that call. If the transaction was a cryptocurrency transfer, then the record includes the TransferList which gives the details of that transfer. If the transaction didn't return anything that should be in the record, then the results field will be set to nothing.

```

| Field | Type |
Description
|
| ----- | ----- |
-----
-----
-----
-----
-----
-----
|
| header | QueryHeader | Standard info sent
from client to node, including the signed payment, and what kind of response is
requested (cost, state proof, both, or neither).
|
| transactionID | TransactionID | The ID of the transaction for which
the record is requested.
|
| includeDuplicates | bool |
Whether records of processing duplicate transactions should be returned along
with the record of processing the first consensus transaction with the given id
whose status was neither INVALID\NODE\ACCOUNT nor INVALID\PAYER\SIGNATURE; or,
if no such record exists, the record of processing the first transaction to
reach consensus with the given transaction id. |
| includechildrecords | bool |
Whether the response should include the records of any child transactions
spawned by the \ top-level transaction with the given transactionID.
|

```

TransactionGetRecordResponse

Response when the client sends the node TransactionGetRecordQuery

Field	Type
Description	

header	ResponseHeader
response from node to client, including the requested fields: cost, or state proof, or both, or neither.	
transactionRecord	TransactionRecord
record	
The requested	
duplicateTransactionRecords	repeated TransactionRecord
The records of processing all consensus transaction with the same id as the distinguished record above, in chronological order.	
childtransactionrecords	repeated TransactionRecord
The records of processing all child transaction spawned by the transaction with the given \ top-level id, in consensus order. Always empty if the top-level status is UNKNOWN.	

transactionreceipt.md:

TransactionReceipt

TransactionReceipt

The summary of a transaction's result so far. If the transaction has not reached consensus, this result will be necessarily incomplete.

Field	Type
Description	

status	ResponseCodeEnum
of the transaction; is UNKNOWN if consensus has not been reached, or if the associated transaction did not have a valid payer signature	
accountID	AccountID
CryptoCreate, the id of the newly created account	
fileID	FileID
FileCreate, the id of the newly created file	
contractID	ContractID
ContractCreate, the id of the newly created contract	
exchangeRate	ExchangeRateSet
in effect when the transaction reached consensus	
topicID	TopicID
ConsensusCreateTopic, the id of the newly created topic.	
topicSequenceNumber	uint64

bytes)</p><p>2. The topicRunningHashVersion below (8 bytes)</p><p>3. The payer account's shard (8 bytes)</p><p>4. The payer account's realm (8 bytes)</p><p>5. The payer account's number (8 bytes)</p><p>6. The topic's shard (8 bytes)</p><p>7. The topic's realm (8 bytes)</p><p>8. The topic's number (8 bytes)</p><p>9. The number of seconds since the epoch before the ConsensusSubmitMessage reached consensus (8 bytes)</p><p>10. The number of nanoseconds since 9. before the ConsensusSubmitMessage reached consensus (4 bytes)</p><p>11. The topicSequenceNumber from above (8 bytes)</p><p>12. The output of the SHA-384 digest of the message bytes from the consensusSubmitMessage (48 bytes)</p> |

transactionrecord.md:

TransactionRecord

repeated TokenAllowanceTransactionRecord

Response when the client sends the node TransactionGetRecordResponse

Field	Type
Label	Description
-----	-----
receipt	TransactionReceipt
The status (reach consensus, or failed, or is unknown) and the ID of any new account/file/instance created.	
transactionHash	bytes
The hash of the Transaction that executed (not the hash of any Transaction that failed for having a duplicate TransactionID)	
consensusTimestamp	Timestamp
The consensus timestamp (or null if didn't reach consensus yet)	
transactionID	TransactionID
The ID of the transaction this record represents	
memo	string
The memo that was submitted as part of the transaction (max 100 bytes)	
transactionFee	uint64
The actual transaction fee charged, not the original transactionFee value from TransactionBody	
contractCallResult	ContractFunctionResult
Record of the value returned by the smart contract function (if it completed and didn't fail) from ContractCallTransaction	
contractCreateResult	ContractFunctionResult
Record of the value returned by the smart contract constructor (if it completed and didn't fail) from ContractCreateTransaction	
transferList	TransferList
All hbar transfers as a result of this transaction, such as fees, or transfers performed by the transaction, or by a smart contract it calls, or by the creation of threshold records that it triggers.	
tokenTransferList	TokenTransferList
	repeated All

Token transfers as a result of this transaction

	scheduleRef		ScheduleID	
	Reference to the scheduled transaction ID that this transaction record represent			
	assessedCustomFees		AssessedCustomFee	repeated All custom fees that were assessed during a CryptoTransfer, and must be paid if the transaction status resolved to SUCCESS
	automatictokenassociations		TokenAssociation	repeated All token associations implicitly created while handling this transaction
	parentconsensustimestamp		Timestamp	
	In the record of an internal transaction, the consensus timestamp of the user transaction that spawned it.			
	alias		bytes	
	In the record of an internal CryptoCreate transaction triggered by a user transaction with a (previously unused) alias, the new account's alias.			

transactionresponse.md:

TransactionResponse

When the client sends the node a transaction of any kind, the node replies with this, which simply says that the transaction passed the precheck (so the node will submit it to the network) or it failed (so it won't). To learn the consensus result, the client should later obtain a receipt (free), or can buy a more detailed record (not free).

Field	Type	Description
nodeId	AccountId	The node ID.
transactionHash	byte	The transaction hash.
transactionId	TransactionId	The transaction ID.

uncheckedsubmit.md:

UncheckedSubmit

Submit an arbitrary (serialized) Transaction to the network without prechecks. Requires superuser privileges.

UncheckedSubmitBody

Field	Type	Description
transactionBytes	bytes	The serialized bytes of the Transaction to be submitted without prechecks

fileappend.md:

FileAppend

FileAppendTransactionBody

Append the given contents to the end of the specified file. If a file is too big to create with a single FileCreateTransaction, then it can be created with the first part of its contents, and then appended as many times as necessary to create the entire file.

Field	Type	Description
-----	-----	-----
fileID	FileID	The file to which the bytes will be appended
contents	bytes	The bytes that will be appended to the end of the specified file

filecreate.md:

FileCreate

FileCreateTransactionBody

Create a new file, containing the given contents. After the file is created, the FileID for it can be found in the receipt, or record, or retrieved with a GetByKey query. The file contains the specified contents (possibly empty). The file will automatically disappear at the expirationTime, unless its expiration is extended by another transaction before that time. If the file is deleted, then its contents will become empty and it will be marked as deleted until it expires, and then it will cease to exist.

The keys field is a list of keys. All the keys on the list must sign to create or modify a file, but only one of them needs to sign in order to delete the file. Each of those "keys" may itself be threshold key containing other keys (including other threshold keys). In other words, the behavior is an AND for create/modify, OR for delete. This is useful for acting as a revocation server. If it is desired to have the behavior be AND for all 3 operations (or OR for all 3), then the list should have only a single Key, which is a threshold key, with N=1 for OR, N=M for AND. If a file is created without ANY keys in the keys field, the file is immutable and ONLY the expirationTime of the file can be changed with a FileUpdate transaction. The file contents or its keys cannot be changed.

An entity (account, file, or smart contract instance) must be created in a particular realm. If the realmID is left null, then a new realm will be created with the given admin key. If a new realm has a null adminKey, then anyone can create/modify/delete entities in that realm. But if an admin key is given, then any transaction to create/modify/delete an entity in that realm must be signed by that key, though anyone can still call functions on smart contract instances that exist in that realm. A realm ceases to exist when everything within it has expired and no longer exists. The current API ignores shardID, realmID, and newRealmAdminKey, and creates everything in shard 0 and realm 0, with a null key. Future versions of the API will support multiple realms and multiple shards.

Field	Type	Description
-----	-----	-----
expirationTime	Timestamp	The time at which this file should expire (unless FileUpdateTransaction is used before then to extend its life)

keys	KeyList	All these keys must sign to create or modify the file. Any one of them can sign to delete the file.
contents	Content	The bytes that are the contents of the file
shardID	ShardID	Shard in which this file is created
realmID	RealmID	The Realm in which to the file is created (leave this null to create a new realm)
newRealmAdminKey	Key	If realmID is null, then this the admin key for the new realm that will be created
memo	string	The memo associated with the file (UTF-8 encoding max 100 bytes)

filedelete.md:

FileDelete

FileDeleteTransactionBody

Delete the given file. After deletion, it will be marked as deleted and will have no contents. But information about it will continue to exist until it expires. A list of keys was given when the file was created. All the keys on that list must sign transactions to create or modify the file, but any single one of them can be used to delete the file. Each "key" on that list may itself be a threshold key containing other keys (including other threshold keys).

Field	Type	Description
fileID	FileID	The file to delete. It will be marked as deleted until it expires. Then it will disappear.

filegetcontents.md:

FileGetContents

FileGetContentsQuery

Get the contents of a file. The content field is empty (no bytes) if the file is empty.

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
fileID	FileID	The file ID of the file whose contents are requested

FileGetContentsResponse

Response when the client sends the node FileGetContentsQuery

Field	Type	Description

header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
fileContents	FileGetContentsResponse.FileContents	the file ID and contents (a state proof can be generated for this)

FileGetContentsResponse

Response when the client sends the node FileGetContentsQuery

Field	Type	Description
-----	-----	
fileID	FileID	The file ID of the file whose contents are being returned
contents	FileContents	The bytes contained in the file

filegetinfo.md:

FileGetInfo

FileGetInfoQuery

Get all of the information about a file, except for its contents. When a file expires, it no longer exists, and there will be no info about it, and the fileInfo field will be blank. If a transaction or smart contract deletes the file, but it has not yet expired, then the fileInfo field will be non-empty, the deleted field will be true, its size will be 0, and its contents will be empty. Note that each file has a FileID, but does not have a filename.

Field	Type	Description

header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
fileID	FileID	The file ID of the file for which information is requested

FileGetInfoResponse

Response when the client sends the node FileGetInfoQuery

Field	Type	Description
-------	------	-------------

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
fileInfo	FileGetInfoResponse.FileInfo	The information about the file (a state proof can be generated for this)

FileGetInfoResponse

Response when the client sends the node FileGetInfoQuery

Field	Type	Description
fileID	FileID	The file ID of the file for which information is requested
size	int64	Number of bytes in contents
expirationTime	Timestamp	The current time at which this account is set to expire
deleted	bool	True if deleted but not yet expired
keys	KeyList	One of these keys must sign in order to modify or delete the file
memo	string	The memo associated with the file (UTF-8 encoding max 100 bytes)
ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs

fileservice.md:

FileService

RPC	Request	Response
Comments		
createFile	Transaction	TransactionResponse
updateFile	Transaction	TransactionResponse
deleteFile	Transaction	TransactionResponse
appendContent	Transaction	TransactionResponse
getFileContent	Query	Response
getFileInfo	Query	Response

systemDelete	Transaction	TransactionResponse	Deletes a file if the submitting account has network admin privileges
systemUndelete	Transaction	TransactionResponse	Undeletes a file if the submitting account has network admin privileges

fileupdate.md:

FileUpdate

FileUpdateTransactionBody

Modify the metadata and/or contents of a file. If a field is not set in the transaction body, the corresponding file attribute will be unchanged. This transaction must be signed by all the keys in the key list of the file being updated. If the keys themselves are being update, then the transaction must also be signed by all the new keys.

Field	Type	Description
fileID	FileID	The ID of the file to update
expirationTime	Timestamp	The new expiry time (ignored if not later than the current expiry)
keys	KeyList	The new list of keys that can modify or delete the file
contents	bytes	The new contents that should overwrite the file's current contents
memo	string	The memo associated with the file (UTF-8 encoding max 100 bytes)

README.md:

File Service

contractcall.md:

ContractCall

ContractCallTransactionBody

Call a function of the given smart contract instance, giving it functionParameters as its inputs. The call can use at maximum the given amount of gas - the paying account will not be charged for any unspent gas.

If this function results in data being stored, an amount of gas is calculated that reflects this storage burden.

The amount of gas used, as well as other attributes of the transaction, e.g. size, number of signatures to be verified, determine the fee for the transaction - which is charged to the paying account.

Field	Type	Description
contractID	ContractID	The contract instance to call
gas	int64	The maximum

amount of gas to use for the call		
amount	int64	Number of
tinybars sent (the function must be payable if this is nonzero)		
functionParameters	bytes	Which
function to call, and the parameters to pass to the function		

contractcalllocal.md:

ContractCallLocal

ContractCallLocalQuery

Call a function of the given smart contract instance, giving it functionParameters as its inputs. It will consume the entire given amount of gas.

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither). The payment must cover the fees and all of the gas offered.
contractID	ContractID	The contract to make a static call against
gas	int64	The amount of gas to use for the call. All of the gas offered will be charged for.
functionParameters	bytes	Which function to call, and the parameters to pass to the function
maxResultSize	int64	Max number of bytes that the result might include. The run will fail if it would have returned more than this number of bytes. \[deprecated 0.20.0]

ContractCallLocalResponse

Response when the client sends the node ContractCallLocalQuery

Field	Type	Description
header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
functionResult	ContractFunctionResult	the value returned by the function (if it completed and didn't fail)

ContractFunctionResult

The result returned by a call to a smart contract function. This is part of the response to a ContractCallLocal query, and is in the record for a ContractCall or ContractCreateInstance transaction. The ContractCreateInstance transaction record has the results of the call to the constructor.

Field	Type	Description
contractID	ContractID	The smart contract instance whose function was called
contractCallResult	bytes	The result returned by the function
errorMessage	string	The message in case there was an error during smart contract execution
bloom	bytes	Bloom filter for record
gasUsed	uint64	Units of gas used to execute contract
logInfo	ContractLoginfo	The log info for events returned by the function
createdContractIDs	repeated ContractID	The list of smart contracts that were created by the function call \[deprecated]
stateChanges	repeated ContractStateChange	The list of state reads and changes caused by this function call
evmaddress	bytes	<p>The 20-byte EVM address of the contract to call. Only populated after release 0.23, where each created contract will have its own record. There will be separate child record for each created contract. Every contract has an EVM address determined by its shard.realm.num id. This address is as follows:</p> <p>The first 4 bytes are the big-endian representation of the shard.</p> <p>The next 8 bytes are the big-endian representation of the realm.</p> <p>The final 8 bytes are the big-endian representation of the number.</p> <p>Contracts created via CREATE2 have an additional, primary address that is derived from the EIP-1014 specification, and does not have a simple relation to a shard.realm.num id. (Please do note that CREATE2 contracts can also be referenced by the three-part EVM address described above.)</p>
gas	int64	The amount of gas available for the call, aka the gasLimit. This field should only be populated when the paired TransactionBody in the record stream is not a ContractCreateTransactionBody or a ContractCallTransactionBody.
amount	int64	Number of tinybars sent (the function must be payable if this is nonzero). This field should only be populated when the paired TransactionBody in the

record stream is not a `ContractCreateTransactionBody` or a `ContractCallTransactionBody`.

	functionParameters	bytes
	The parameters passed into the contract call. This field should only be populated when the paired <code>TransactionBody</code> in the record stream is not a <code>ContractCreateTransactionBody</code> or a <code>ContractCallTransactionBody</code> .	

	signerNonce	int64
	The value of the signer account nonce. This value can be null.	

ContractLoginInfo

The log information for an event returned by a smart contract function call. One function call may return several such events.

Field	Type	Description
contractID	ContractID	Address of a contract that emitted the event
bloom	bytes	Bloom filter for a particular log
topic	repeated bytes	Topics of a particular event
data	bytes	Event data

ContractStateChange

The storage changes to a smart contract's storage as a side effect of the function call.

Field	Type	Description
contractID	ContractID	The contract to which the storage changes apply to
storageChanges	repeated StorageChange	The list of storage changes

StorageChange

A storage slot change description.

Field	Type	Description
slot	bytes	The storage slot changed. Up to 32 bytes, big-endian, zero bytes left trimmed
valueRead	bytes	The value read from the storage slot. Up to 32 bytes, big-endian, zero bytes left trimmed. Because of the way <code>SSTORE</code> operations are charged the slot is always read before being written to
valueWritten	google.protobuf.BytesValue	The new value written to the slot. Up to 32 bytes, big-endian, zero bytes left trimmed. If a value of zero is

written the valueWritten will be present but the inner value will be absent. If a value was read and not written this value will not be present. |

contractcreate.md:

ContractCreate

ContractCreateTransactionBody

Start a new smart contract instance. After the instance is created, the ContractID for it is in the receipt, or can be retrieved with a GetByKey query, or by asking for a Record of the transaction to be created, and retrieving that. The instance will run the bytecode stored in the given file, referenced either by FileID or by the transaction ID of the transaction that created the file. The constructor will be executed using the given amount of gas, and any unspent gas will be refunded to the paying account. Constructor inputs come from the given constructorParameters.

Field	Type	Description
fileID	FileID	the file containing the smart contract byte code. A copy will be made and held by the contract instance, and have the same expiration time as the instance. The file is referenced one of two ways:
adminKey	Key	the state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
gas	int64	gas to run the constructor
initialBalance	int64	initial number of tinybars to put into the cryptocurrency account associated with and owned by the smart contract
proxyAccountID	AccountID	ID of the account to which this account is proxy staked. If proxyAccountID is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking, or if it is not currently running a node, then it will behave as if proxyAccountID was null.
autoRenewPeriod	Duration	the instance will charge its account every this many seconds to renew for this long
constructorParameters	bytes	parameters to pass to the constructor
shardID	ShardID	shard in which to create this
realmID	RealmID	realm in which to create this (leave this null to create a new realm)

newRealmAdminKey	Key	if realmID is null, then this the
admin key for the new realm that will be created		
memo	string	the memo
that was submitted as part of the contract (max 100 bytes)		

###

contractdelete.md:

ContractDelete

ContractDeleteTransactionBody

Delete a smart contract.

Field	Type	Description
contractID	ContractID	The Contract ID instance to delete (this can't be changed)
obtainers		one of:
transferAccountID	AccountID	The account ID which will receive all remaining hbars
transferContractID	ContractID	The contract ID which will receive all remaining hbars

contractgetbytecode.md:

ContractGetByteCode

ContractGetBytecodeQuery

Get the bytecode for a smart contract instance

Field	Type	Description
header	QueryHeader	standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	the contract for which information is requested

ContractGetBytecodeResponse

Response when the client sends the node ContractGetBytecodeQuery

Field	Type	Description
-------	------	-------------

header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither	
bytecode	bytes	the bytecode	

contractgetinfo.md:

ContractGetInfo

ContractGetInfoQuery

Get information about a smart contract instance. This includes the account that it uses, the file containing its bytecode, and the time when it will expire.

Field	Type	Description
-----	-----	-----
header	QueryHeader	standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	the contract for which information is requested

ContractGetInfoResponse

Response when the client sends the node ContractGetInfoQuery

Field	Type	Description
-----	-----	-----
header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
contractInfo	ContractGetInfoResponse.ContractInfo	the information about this contract instance (a state proof can be generated for this)

ContractGetInfoResponse.ContractInfo

Response when the client sends the node ContractGetInfoQuery

Field	Type	Description
-----	-----	-----
contractID	ContractID	ID of the contract instance, in the format used in transactions
accountID	AccountID	ID of the

cryptocurrency account owned by the contract instance, in the format used in transactions

contractAccountID	String	ID of both the contract instance and the cryptocurrency account owned by the contract instance, in the format used by Solidity
adminKey	Key	The state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
expirationTime	Timestamp	The current time at which this contract instance (and its account) is set to expire
autoRenewPeriod	Duration	The expiration time will extend every this many seconds. If there are insufficient funds, then it extends as long as possible. If the account is empty when it expires, then it is deleted.
storage	uint64	Number of bytes of storage being used by this instance (which affects the cost to extend the expiration time)
memo	String	The memo associated with the contract (max 100 bytes)
balance	uint64	The current balance, in tinybars
deleted	bool	Whether the contract has been deleted
tokenRelationships	repeated TokenRelationship	The tokens associated to the contract
ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs

contractgetrecords.md:

ContractGetRecords

ContractGetRecordsQuery

Get all the records for a smart contract instance, for any function call (or the constructor call) during the last 25 hours, for which a Record was requested.

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	The smart contract instance for which the records should be retrieved

ContractGetRecordsResponse

Response when the client sends the node ContractGetRecordsQuery

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
contractID	ContractID	The smart contract instance that this record is for
records	TransactionRecord	List of records, each with contractCreateResult or contractCallResult as its body

contractupdate.md:

ContractUpdate

ContractUpdateTransactionBody

Modify a smart contract instance to have the given parameter values. Any null field is ignored (left unchanged). If only the `contractInstanceExpirationTime` is being modified, then no signature is needed on this transaction other than for the account paying for the transaction itself. But if any of the other fields are being modified, then it must be signed by the `adminKey`. The use of `adminKey` is not currently supported in this API, but in the future will be implemented to allow these fields to be modified, and also to make modifications to the state of the instance. If the contract is created with no admin key, then none of the fields can be changed that need an admin signature, and therefore no admin key can ever be added. The `adminKey` can be used to add flexibility to the management of smart contract behavior, but this is optional. If the smart contract is created without an `adminKey`, then such a key can never be added, and none of the fields can be changed that need an admin signature.

Field	Type	Description
contractID	ContractID	The Contract ID instance to update (this can't be changed)
expirationTime	Timestamp	Extend the expiration of the instance and its account to this time (no effect if it already is this time or later)
adminKey	Key	The state of the instance can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance.
proxyAccountID	AccountID	ID of the account to which this account is proxy staked. If <code>proxyAccountID</code> is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a

node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking , or if it is not currently running a node, then it will behave as if proxyAccountID was null. |

autoRenewPeriod	Duration	The instance will charge its account every this many seconds to renew for this long
-----------------	----------	---

|

fileID	FileID	The file ID of file containing the smart contract byte code. A copy will be made and held by the contract instance, and have the same expiration time as the instance. \[deprecated 0.20.0]
--------	--------	---

|

memo	string	The memo associated with the contract (max 100 bytes)
------	--------	---

|

README.md:

Smart Contracts

smartcontractservice.md:

SmartContractService

RPC	Request	Response	Comments
-----	-----	-----	-----
createContract	Transaction	TransactionResponse	Creates a contract
updateContract	Transaction	TransactionResponse	Updates a contract with the content
contractCallMethod	Transaction	TransactionResponse	Calls a contract
getContractInfo	Query	Response	Retrieves the contract information
contractCallLocalMethod	Query	Response	Calls a smart contract to be run on a single node
ContractGetBytecode	Query	Response	Retrieves the byte code of a contract
getBySolidityID	Query	Response	Retrieves a contract by its Solidity address
getTxRecordByContractID	Query	Response	Retrieves the 25-hour records stored for a contract
deleteContract	Transaction	TransactionResponse	Deletes a contract instance and transfers any remaining hbars to a specified receiver
systemDelete	Transaction	TransactionResponse	Deletes a contract if the submitting account has network admin privileges
systemUndelete	Transaction	TransactionResponse	Undeletes a contract if the submitting account has network admin privileges

README.md:

Schedule Service

schedulabletransactionbody.md:

SchedulableTransactionBody

A schedulable transaction. Note that the global/dynamic system property `scheduling.whitelist` controls which transaction types may be scheduled. In Hedera Services 0.13.0, it will include only `CryptoTransfer` and `ConsensusSubmitMessage` functions.

Field Description	Type	
-----	-----	-----
transactionFee	uint64	The maximum transaction fee the client is willing to pay
memo	string	A memo to include the execution record; the UTF-8 encoding may be up to 100 bytes and must not include the zero byte
Oneof Data:		
contractCall	ContractCallTransactionBody	Calls a function of a contract instance
contractCreateInstance	ContractCreateTransactionBody	Creates a contract instance
contractUpdateInstance	ContractUpdateTransactionBody	Updates a contract
contractDeleteInstance	ContractDeleteTransactionBody	Delete contract and transfer remaining balance into specified account
cryptoApproveAllowance	CryptoApproveAllowanceTransactionBody	Adds one or more approved allowances for spenders to transfer the paying account's hbar or tokens.
cryptoDeleteAllowance	CryptoDeleteAllowanceTransactionBody	Deletes one or more NFT allowances from an owner's account
cryptoCreateAccount	CryptoCreateTransactionBody	Create a new cryptocurrency account
cryptoDelete	CryptoDeleteTransactionBody	Delete a cryptocurrency account (mark as deleted, and transfer hbars out)
cryptoTransfer	CryptoTransferTransactionBody	Transfer amount between accounts
cryptoUpdateAccount	CryptoUpdateTransactionBody	Modify information such as the expiration date for an account
fileAppend	FileAppendTransactionBody	Add bytes to the end of the contents of a file
fileCreate	FileCreateTransactionBody	Create a new file
fileDelete	FileDeleteTransactionBody	Delete a file (remove contents and mark as deleted until it expires)

fileUpdate	FileUpdateTransactionBody	Modify
information such as the expiration date for a file		
systemDelete	SystemDeleteTransactionBody	Hedera
administrative deletion of a file or smart contract		
systemUndelete	SystemUndeleteTransactionBody	To
undelete an entity deleted by SystemDelete		
freeze	FreezeTransactionBody	Freeze the
nodes		
consensusCreateTopic	ConsensusCreateTopicTransactionBody	Creates a
topic		
consensusUpdateTopic	ConsensusUpdateTopicTransactionBody	Updates a
topic		
consensusDeleteTopic	ConsensusDeleteTopicTransactionBody	Deletes a
topic		
consensusSubmitMessage	ConsensusSubmitMessageTransactionBody	Submits
message to a topic		
tokenCreation	TokenCreateTransactionBody	Creates a
token instance		
tokenFreeze	TokenFreezeAccountTransactionBody	Freezes
account not to be able to transact with a token		
tokenUnfreeze	TokenUnfreezeAccountTransactionBody	Unfreezes
account for a token		
tokenGrantKyc	TokenGrantKycTransactionBody	Grants KYC
to an account for a token		
tokenRevokeKyc	TokenRevokeKycTransactionBody	Revokes
KYC of an account for a token		
tokenDeletion	TokenDeleteTransactionBody	Deletes a
token instance		
tokenUpdate	TokenUpdateTransactionBody	Updates a
token instance		
tokenMint	TokenMintTransactionBody	Mints new
tokens to a token's treasury account		
tokenBurn	TokenBurnTransactionBody	Burns
tokens from a token's treasury account		
tokenWipe	TokenWipeAccountTransactionBody	Wipes
amount of tokens from an account		
tokenAssociate	TokenAssociateTransactionBody	Associate
tokens to an account		
tokenDissociate	TokenDissociateTransactionBody	Dissociate
tokens from an account		
tokenpause	TokenPauseTransactionBody	Pauses the
Token		

tokenunpause the Token	TokenUnpauseTransactionBody	Unpauses
scheduleDelete	ScheduleDeleteTransactionBody	Marks a schedule in the network's action queue as deleted, preventing it from executing
tokenfeescheduleupdate	TokenFeeScheduleUpdateTransactionBody	Updates a token's custom fee schedule
cryptoAdjustAllowance	CryptoAdjustAllowanceTransactionBody	Adjusts the approved allowance for a spender to transfer the paying account's hbar or tokens
cryptoApproveAllowance	CryptoApproveAllowanceTransactionBody	Adds one or more approved allowances for spenders to transfer the paying account's hbar or tokens

schedulecreate.md:

ScheduleCreate

Create a new schedule entity (or simply, `<i>schedule</i>`) in the network's action queue. Upon SUCCESS, the receipt contains the `\ScheduleID\` of the created schedule. A schedule entity includes a `ScheduledTransactionBody` to be executed when the schedule has collected enough signing Ed25519 keys to satisfy the scheduled transaction's signing requirements. Upon `\SUCCESS\`, the receipt also includes the `ScheduledTransactionID` to use to query for the record of the scheduled transaction's execution (if it occurs).

The expiration time of a schedule is always 30 minutes; it remains in state and can be queried using `GetScheduleInfo` until expiration, no matter if the scheduled transaction has executed or marked deleted.

If the `adminKey` field is omitted, the resulting schedule is immutable. If the `adminKey` is set, the `ScheduleDelete` transaction can be used to mark it as deleted. The creator may also specify an optional memo whose UTF-8 encoding is at most 100 bytes and does not include the zero byte is also supported.

When a scheduled transaction whose schedule has collected enough signing keys is executed, the network only charges its payer the service fee, and not the node and network fees. If the optional `payerAccountID` is set, the network charges this account. Otherwise it charges the payer of the originating `ScheduleCreate`.

Two `ScheduleCreate` transactions are identical if they are equal in all their fields other than `payerAccountID`. (Here "equal" should be understood in the sense of gRPC object equality in the network software runtime. In particular, a gRPC object with unknown fields is not equal to a gRPC object without unknown fields, even if they agree on all known fields.)

A `ScheduleCreate` transaction that attempts to re-create an identical schedule already in state will receive a receipt with status

`IDENTICAL\SCHEDULE\ALREADY\CREATED`; the receipt will include the `ScheduleID` of the extant schedule, which may be used in a subsequent `ScheduleSign` transaction. (The receipt will also include the `TransactionID` to use in querying or the receipt or record of the scheduled transaction.)

Other notable response codes include, `INVALID\ACCOUNT\ID`, `UNSCHEDULABLE\TRANSACTION`, `UNRESOLVABLE\REQUIRED\SIGNER`, `INVALID\SIGNATURE`.

ScheduleCreateTransactionBody

Field Description	Type	
----------------------	------	--

Field	Type	Description
SchedulableTransactionBody	ScheduledTransactionBody	The scheduled transaction
adminKey	Key	An optional Hedera key which can be used to sign a ScheduleDelete and remove the schedule
payerAccountId	AccountID	An optional id of the account to be charged the service fee for the scheduled transaction at the consensus time that it executes (if ever); defaults to the ScheduleCreate payer if not give
memo	string	An optional memo with a UTF-8 encoding of no more than 100 bytes which does not contain the zero byte

scheduledelete.md:

ScheduleDelete

Marks a schedule in the network's action queue as deleted. Must be signed by the admin key of the target schedule. A deleted schedule cannot receive any additional signing keys, nor will it be executed.

Other notable response codes include, INVALID\SCHEDULE\ID, SCHEDULE\WAS\DELETED, SCHEDULE\WAS\EXECUTED, SCHEDULE\IS\IMMUTABLE. For more information please see the section of this documentation on the ResponseCode enum.

ScheduleDeleteTransactionBody

Field	Type	Description
scheduleID	ScheduleID	The ID of the Scheduled Entity

schedulegetinfo.md:

ScheduleGetInfo

Gets information about a schedule in the network's action queue. Responds with INVALID\SCHEDULE\ID if the requested schedule doesn't exist.

ScheduleGetInfoQuery

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
scheduleID	ScheduleID	The ID of the Scheduled Entity

ScheduleInfo

Information summarizing schedule state.

Field	Type	Description

scheduleID	ScheduleID	The id of the schedule
creatorAccountID	AccountID	The id of the account responsible for the service fee of the scheduled transaction
payerAccountIDAccountID	AccountID	The account which is going to pay for the execution of the scheduled transaction
scheduledTransactionBody	SchedulableTransactionBody	The scheduled transaction
signers	KeyList	The Ed25519 keys the network deems to have signed the scheduled transaction
adminKey	Key	The key used to delete the schedule from state
memo	string	The publicly visible memo of the schedule
expirationTime	TimeStamp	The epoch second at which the schedule will expire
scheduledTransactionID	TransactionID	The transaction id that will be used in the record of the scheduled transaction (if it executes)
ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs
oneof data :		
deletionTime	TimeStamp	If the schedule has been deleted, the consensus time when this occurred
executionTime	TimeStamp	If the schedule has been executed, the consensus time when this occurred

ScheduleGetInfoResponse

Response wrapper for the ScheduleInfo.

Field	Type	Description

header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither.

scheduleInfo ScheduleInfo	The information requested about this schedule instance	
-----------------------------	--	--

scheduleservice.md:

ScheduleService

Transactions and queries for the Schedule Service

The Schedule Service allows transactions to be submitted without all the required signatures and allows anyone to provide the required signatures independently after a transaction has already been created.

Execution:

Scheduled Transactions are executed once all required signatures are collected and witnessed. Every time new signature is provided, a check is performed on the "readiness" of the execution.

The Scheduled Transaction will be executed immediately after the transaction that triggered it and will be externalized in a separate Transaction Record.

Transaction Record:

The timestamp of the Scheduled Transaction will be equal to consensusTimestamp + 1 nano, where consensusTimestamp is the timestamp of the transaction that triggered the execution.

The Transaction ID of the Scheduled Transaction will have the scheduled property set to true and inherit the transactionValidStart and accountID from the ScheduleCreate transaction.

The scheduleRef property of the transaction record will be populated with the ScheduleID of the Scheduled Transaction.

Post execution:

Once a given Scheduled Transaction executes, it will be removed from the ledger and any upcoming operation referring the ScheduleID will resolve to INVALID\SCHEDULE\ID.

Expiry:

Scheduled Transactions have a global expiry time txExpiryTimeSecs (Currently set to 30 minutes). If txExpiryTimeSecs pass and the Scheduled Transaction haven't yet executed, it will be removed from the ledger as if ScheduleDelete operation is executed.

ScheduleService

RPC	Request	Response
Comments		
-----	-----	-----
createSchedule	Transaction	TransactionResponse
by submitting the transaction		
signSchedule	Transaction	TransactionResponse
submitting the transaction		
deleteSchedule	Transaction	TransactionResponse
by submitting the transaction		
getScheduleInfo	Query	Response
Retrieves the metadata of a schedule entity		

schedulesign.md:

ScheduleSign

Adds zero or more signing keys to a schedule. If the resulting set of signing keys satisfy the scheduled transaction's signing requirements, it will be executed immediately after the triggering ScheduleSign.

Upon SUCCESS, the receipt includes the scheduledTransactionID to use to query for the record of the scheduled transaction's execution (if it occurs).

Other notable response codes include INVALID\SCHEDULE\ID, SCHEDULE\WAS\DELETED, INVALID\ACCOUNT\ID, UNRESOLVABLE\REQUIRED\SIGNERS, SOME\SIGNATURES\WERE\INVALID, and NO\NEW\VALID\SIGNATURES. For more information please see the section of this documentation on the ResponseCode enum.

ScheduleSignTransactionBody

Field	Type	Description
-----	-----	-----
scheduleID	ScheduleID	The ID of the Scheduled Entity

contractcall.md:

ContractCall

ContractCallTransactionBody

Call a function of the given smart contract instance, giving it functionParameters as its inputs. The call can use at maximum the given amount of gas - the paying account will not be charged for any unspent gas.\

If this function results in data being stored, an amount of gas is calculated that reflects this storage burden.\

The amount of gas used, as well as other attributes of the transaction, e.g. size, number of signatures to be verified, determine the fee for the transaction - which is charged to the paying account.

Field	Type	Description
-----	-----	-----
contractID	ContractID	The contract instance to call
gas	int64	The maximum amount of gas to use for the call
amount	int64	Number of tinybars sent (the function must be payable if this is nonzero)
functionParameters	bytes	Which function to call, and the parameters to pass to the function

contractcalllocal.md:

ContractCallLocal

ContractCallLocalQuery

Call a function of the given smart contract instance, giving it functionParameters as its inputs. It will consume the entire given amount of gas.

Field	Type	
Description		
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither). The payment must cover the fees and all of the gas offered.
contractID	ContractID	The contract to make a static call against
gas	int64	The amount of gas to use for the call. All of the gas offered will be charged for.
functionParameters	bytes	Which function to call, and the parameters to pass to the function
maxResultSize	int64	Max number of bytes that the result might include. The run will fail if it would have returned more than this number of bytes. \[deprecated 0.20.0]

ContractCallLocalResponse

Response when the client sends the node ContractCallLocalQuery

Field	Type	Description
header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
functionResult	ContractFunctionResult	the value returned by the function (if it completed and didn't fail)

ContractFunctionResult

The result returned by a call to a smart contract function. This is part of the response to a `ContractCallLocal` query, and is in the record for a `ContractCall` or `ContractCreateInstance` transaction. The `ContractCreateInstance` transaction record has the results of the call to the constructor.

[illegible]

```

-----
-----
-----
-----
|
| contractID          | ContractID          | The smart
contract instance whose function was called
|
| contractCallResult | bytes
| The result returned by the function
|
| errorMessage       | string
| The message in case there was an error during smart contract execution
|
| bloom              | bytes
| Bloom filter for record
|
| gasUsed            | uint64
| Units of gas used to execute contract
|
| logInfo            | ContractLoginInfo   | The log info for
events returned by the function
|
| createdContractIDs | repeated ContractID | The list of
smart contracts that were created by the function call \[deprecated]
|
| stateChanges       | repeated ContractStateChange | The list of state reads
and changes caused by this function call
|
| evmaddress         | bytes
| <p>The 20-byte EVM address of the contract to call. Only populated after
release 0.23, where each created contract will have its own record. There will
be separate child record for each created contract.<br><br>Every contract has an
EVM address determined by its shard.realm.num id. This address is as
follows:<br><br>The first 4 bytes are the big-endian representation of the
shard.<br><br>The next 8 bytes are the big-endian representation of the
realm.<br><br>The final 8 bytes are the big-endian representation of the
number.<br><br>Contracts created via CREATE2 have an <strong>additional, primary
address</strong> that is derived from the <a
href="https://eips.ethereum.org/EIPS/eip-1014">EIP-1014</a> specification, and
does not have a simple relation to a shard.realm.num id.<br><br>(Please do note
that CREATE2 contracts can also be referenced by the three-part EVM address
described above.)</p>
|
| gas                | int64
| The amount of gas available for the call, aka the gasLimit. This field should
only be populated when the paired TransactionBody in the record stream is not a
ContractCreateTransactionBody or a ContractCallTransactionBody.
|
| amount             | int64
| Number of tinybars sent (the function must be payable if this is nonzero).
This field should only be populated when the paired TransactionBody in the
record stream is not a ContractCreateTransactionBody or a
ContractCallTransactionBody.
|
| functionParameters | bytes
| The parameters passed into the contract call. This field should only be
populated when the paired TransactionBody in the record stream is not a
ContractCreateTransactionBody or a ContractCallTransactionBody.
|

```

ContractLoginInfo

The log information for an event returned by a smart contract function call. One function call may return several such events.

Field	Type	Description
contractID	ContractID	Address of a contract that emitted the event
bloom	bytes	Bloom filter for a particular log
topic	repeated bytes	Topics of a particular event
data	bytes	Event data

ContractStateChange

The storage changes to a smart contract's storage as a side effect of the function call.

Field	Type	Description
contractID	ContractID	The contract to which the storage changes apply to
storageChanges	repeated StorageChange	The list of storage changes

StorageChange

A storage slot change description.

Field	Type	Description
slot	bytes	The storage slot changed. Up to 32 bytes, big-endian, zero bytes left trimmed
valueRead	bytes	The value read from the storage slot. Up to 32 bytes, big-endian, zero bytes left trimmed. Because of the way SSTORE operations are charged the slot is always read before being written to
valueWritten	google.protobuf.BytesValue	The new value written to the slot. Up to 32 bytes, big-endian, zero bytes left trimmed. If a value of zero is written the valueWritten will be present but the inner value will be absent. If a value was read and not written this value will not be present.

contractcreate.md:

ContractCreate

ContractCreateTransactionBody

Start a new smart contract instance. After the instance is created, the ContractID for it is in the receipt, or can be retrieved with a GetByKey query, or by asking for a Record of the transaction to be created, and retrieving that. The instance will run the bytecode stored in the given file, referenced either by FileID or by the transaction ID of the transaction that created the file. The constructor will be executed using the given amount of gas, and any unspent gas

will be refunded to the paying account. Constructor inputs come from the given constructorParameters.

Field Description	Type	
fileID	FileID	the file containing the smart contract byte code. A copy will be made and held by the contract instance, and have the same expiration time as the instance. The file is referenced one of two ways:
adminKey	Key	the state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
gas	int64	gas to run the constructor
initialBalance	int64	initial number of tinybars to put into the cryptocurrency account associated with and owned by the smart contract
proxyAccountID	AccountID	ID of the account to which this account is proxy staked. If proxyAccountID is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the proxyAccountID account refuses to accept proxy staking, or if it is not currently running a node, then it will behave as if proxyAccountID was null.
autoRenewPeriod	Duration	the instance will charge its account every this many seconds to renew for this long
constructorParameters	bytes	parameters to pass to the constructor
shardID	ShardID	shard in which to create this
realmID	RealmID	realm in which to create this (leave this null to create a new realm)
newRealmAdminKey	Key	if realmID is null, then this the admin key for the new realm that will be created
memo	string	the memo that was submitted as part of the contract (max 100 bytes)

###

contractdelete.md:

ContractDelete

ContractDeleteTransactionBody

Delete a smart contract.

Field	Type	Description
contractID	ContractID	The Contract ID instance to delete (this can't be changed)
obtainers		
one of:		
transferAccountID	AccountID	The account ID which will receive all remaining hbars
transferContractID	ContractID	The contract ID which will receive all remaining hbars

contractgetbytecode.md:

ContractGetByteCode

ContractGetBytecodeQuery

Get the bytecode for a smart contract instance

Field	Type	Description
header	QueryHeader	standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	the contract for which information is requested

ContractGetBytecodeResponse

Response when the client sends the node ContractGetBytecodeQuery

Field	Type	Description
header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
bytecode	bytes	the bytecode

contractgetinfo.md:

ContractGetInfo

ContractGetInfoQuery

Get information about a smart contract instance. This includes the account that it uses, the file containing its bytecode, and the time when it will expire.

Field	Type	Description
-------	------	-------------

Field	Type	Description
header	QueryHeader	standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	the contract for which information is requested

ContractGetInfoResponse

Response when the client sends the node ContractGetInfoQuery

Field	Type	Description
header	ResponseHeader	standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
contractInfo	ContractGetInfoResponse.ContractInfo	the information about this contract instance (a state proof can be generated for this)

ContractGetInfoResponse.ContractInfo

Response when the client sends the node ContractGetInfoQuery

Field	Type	Description
contractID	ContractID	ID of the contract instance, in the format used in transactions
accountID	AccountID	ID of the cryptocurrency account owned by the contract instance, in the format used in transactions
contractAccountID	String	ID of both the contract instance and the cryptocurrency account owned by the contract instance, in the format used by Solidity
adminKey	Key	The state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
expirationTime	Timestamp	The current time at which this contract instance (and its account) is set to expire

autoRenewPeriod	Duration	The expiration time will extend every this many seconds. If there are insufficient funds, then it extends as long as possible. If the account is empty when it expires, then it is deleted.
storage	uint64	Number of bytes of storage being used by this instance (which affects the cost to extend the expiration time)
memo	String	The memo associated with the contract (max 100 bytes)
balance	uint64	The current balance, in tinybars
deleted	bool	Whether the contract has been deleted
tokenRelationships	repeated TokenRelationship	The tokens associated to the contract
ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs

contractgetrecords.md:

ContractGetRecords

ContractGetRecordsQuery

Get all the records for a smart contract instance, for any function call (or the constructor call) during the last 25 hours, for which a Record was requested.

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither).
contractID	ContractID	The smart contract instance for which the records should be retrieved

ContractGetRecordsResponse

Response when the client sends the node ContractGetRecordsQuery

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
contractID	ContractID	The smart contract instance that this record is for

```
| records      | TransactionRecord | List of records, each with
contractCreateResult or contractCallResult as its body
|
```

```
# contractupdate.md:
```

```
ContractUpdate
```

```
ContractUpdateTransactionBody
```

Modify a smart contract instance to have the given parameter values. Any null field is ignored (left unchanged). If only the `contractInstanceExpirationTime` is being modified, then no signature is needed on this transaction other than for the account paying for the transaction itself. But if any of the other fields are being modified, then it must be signed by the `adminKey`. The use of `adminKey` is not currently supported in this API, but in the future will be implemented to allow these fields to be modified, and also to make modifications to the state of the instance. If the contract is created with no admin key, then none of the fields can be changed that need an admin signature, and therefore no admin key can ever be added. The `adminKey` can be used to add flexibility to the management of smart contract behavior, but this is optional. If the smart contract is created without an `adminKey`, then such a key can never be added, and none of the fields can be changed that need an admin signature.

Field	Type	Description
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----
contractID	ContractID	The Contract ID instance to update (this can't be changed)
expirationTime	Timestamp	Extend the expiration of the instance and its account to this time (no effect if it already is this time or later)
adminKey	Key	The state of the instance can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance.
proxyAccountID	AccountID	ID of the account to which this account is proxy staked. If <code>proxyAccountID</code> is null, or is an invalid account, or is an account that isn't a node, then this account is automatically proxy staked to a node chosen by the network, but without earning payments. If the <code>proxyAccountID</code> account refuses to accept proxy staking, or if it is not currently running a node, then it will behave as if <code>proxyAccountID</code> was null.
autoRenewPeriod	Duration	The instance will charge its account every this many seconds to renew for this long
fileID	FileID	The file ID of file containing the smart contract byte code. A copy will be made and held by the contract instance, and have the same expiration time as the instance. \[deprecated 0.20.0]
memo	string	The memo associated with the contract (max 100 bytes)

```
# smartcontractservice.md:
```

SmartContractService

RPC Response	Request	Comments

createContract	Transaction	TransactionResponse Creates a contract
updateContract	Transaction	TransactionResponse Updates a contract with the content
contractCallMethod	Transaction	TransactionResponse Calls a contract
getContractInfo	Query	Response
Retrieves the contract information		
contractCallLocalMethod	Query	Response
Calls a smart contract to be run on a single node		
ContractGetBytecode	Query	Response
Retrieves the byte code of a contract		
getBySolidityID	Query	Response
Retrieves a contract by its Solidity address		
getTxRecordByContractID	Query	Response
Retrieves the 25-hour records stored for a contract		
deleteContract	Transaction	TransactionResponse Deletes a contract instance and transfers any remaining hbars to a specified receiver
systemDelete	Transaction	TransactionResponse Deletes a contract if the submitting account has network admin privileges
systemUndelete	Transaction	TransactionResponse Undeletes a contract if the submitting account has network admin privileges

README.md:

Token Service

tokenassociate.md:

TokenAssociate

Associates the provided account with the provided tokens. Must be signed by the provided Account's key.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALIDTOKENREF.

If any of the provided tokens has been deleted, the transaction will resolve to TOKENWASDELETED.

If an association between the provided account and any of the tokens already exists, the transaction will resolve to `TOKENALREADYASSOCIATEDTOACCOUNT`.

If the provided account's associations count exceeds the constraint of maximum token associations per account, the transaction will resolve to `TOKENSPERACCOUNTLIMITEXCEEDED`.

On success, associations between the provided account and tokens are made and the account is ready to interact with the tokens.

TokenAssociateTransactionBody

Field	Type	Description
-----	-----	

account	AccountID	The account to be associated with the provided tokens
tokens	repeated TokenID	The tokens to be associated with the provided account. In the case of <code>NONFUNGIBLEUNIQUE</code> Type, once an account is associated, it can hold any number of NFTs (serial numbers) of that token type.

tokenburn.md:

TokenBurn

Burns tokens from the Token's treasury Account. If no Supply Key is defined, the transaction will resolve to `TOKENHASNOSUPPLYKEY`.

The operation decreases the Total Supply of the Token. Total supply cannot go below zero.

The amount provided must be in the lowest denomination possible. Example:

Token A has 2 decimals. In order to burn 100 tokens, one must provide amount of 10000. In order to burn 100.55 tokens, one must provide amount of 10055.

TokenBurnTransactionBody

Field	Type	Description
-----	-----	

token	TokenID	The token for which to burn tokens. If token does not exist, transaction results in <code>INVALID\TOKEN\ID</code>
amount	uint64	The amount to burn from the Treasury Account. Amount must be a positive non-zero number, not bigger than the token balance of the treasury account (<code>0; balance]</code> , represented in the lowest denomination.
serialNumbers	repeated int64	Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code> . The list of serial numbers to be burned.

tokencreate.md:

TokenCreate

Create a new token. After the token is created, the Token ID for it is in the receipt.

The specified Treasury Account is receiving the initial supply of tokens as-well as the tokens from the Token Mint operation once executed. The balance of the treasury account is decreased when the Token Burn operation is executed.

The `initialSupply` is the initial supply of the smallest parts of tokens (like a tinybar, not an hbar). The supply that is going to be put in circulation is going to be the `initial` supply provided.

The supply can change over time. If the total supply at some moment is S parts of tokens, and the token is using D decimals, then S must be less than or equal to $2^{63}-1$, which is 9,223,372,036,854,775,807. The number of whole tokens (not parts) will $S / 10^D$.

If decimals is 8 or 11, then the number of whole tokens can be at most a few billions or millions, respectively. For example, it could match Bitcoin (21 million whole tokens with 8 decimals) or hbars (50 billion whole tokens with 8 decimals). It could even match Bitcoin with milli-satoshis (21 million whole tokens with 11 decimals). (ed

Example:

Token A has an initial supply set to 10\000 and decimals set to 2. The tokens that will be put into circulation are going be 100.

Token B has an initial supply set to 10⁰¹²345678 and decimals set to 8. The number of tokens that will be put into circulation are going to be 100.12345678

Creating immutable token: Token can be created as immutable if the adminKey is omitted. In this case, the name, symbol, treasury, management keys, expiry and renew properties cannot be updated. If a token is created as immutable, anyone is able to extend the expiry time by paying the fee.

TokenCreateTransactionBody

Field	Type
Description	
Signature Required	

name	string
The publicly visible name of the token. The token name is specified as a Unicode string. Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0 byte (NUL).	
N/A	
symbol	string
The publicly visible token symbol. The token symbol is specified as a Unicode string. Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0 byte (NUL).	
N/A	
decimals	uint32
For tokens of type FUNGIBLECOMMON - the number of decimal places a token is divisible by. For tokens of type NONFUNGIBLEUNIQUE - value must be 0.	
N/A	
initialSupply	uint64
Specifies the initial supply of tokens to be put in circulation. The initial	

supply is sent to the Treasury Account. The supply is in the lowest denomination possible. Maximum supply of tokens: 9,223,372,036,854,775,807 . In the case for NONFUNGIBLEUNIQUE Type the value must be 0 | N/A

treasury	AccountID
----------	-----------

| The account which will act as a treasury for the token. This account will receive the specified initial supply or the newly minted NFTs in the case for NONFUNGIBLEUNIQUE Type.

Required	
----------	--

adminKey	Key	The key which can perform update/delete operations on the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted)
----------	-----	---

If set, required	
------------------	--

kyckKey	Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
---------	-----	---

If set, required	
------------------	--

freezeKey	Key	The key which can sign to freeze or unfreeze an account for token transactions. If empty, freezing is not possible
-----------	-----	--

If set, required	
------------------	--

wipeKey	Key	The key which can wipe the token balance of an account. If empty, wipe is not possible
---------	-----	--

If set, required	
------------------	--

supplyKey	Key	The key which can change the supply of a token. The key is used to sign Token Mint/Burn operations
-----------	-----	--

If set, required	
------------------	--

freezeDefault	bool	The default Freeze status (frozen or unfrozen) of Hedera accounts relative to this token. If true, an account must be unfrozen before it can receive the token
---------------	------	--

N/A	
-----	--

expiry	uint64	The epoch second at which the token should expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod
--------	--------	--

N/A	
-----	--

autoRenewAccount	AccountID	An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval
------------------	-----------	---

N/A	
-----	--

autoRenewPeriod	uint64	The interval at which the auto-renew account will be charged to extend the token's expiry
-----------------	--------	---

N/A	
-----	--

memo	string	The memo associated with the token (UTF-8 encoding max 100 bytes)
------	--------	---

N/A	
-----	--

tokenType	TokenType	IWA compatibility. Specifies the token type. Defaults to FUNGIBLECOMMON
-----------	-----------	---

N/A	
-----	--

supplyType	TokenSupplyType	IWA compatibility. Specified the token supply type. Defaults to INFINITE
------------	-----------------	--

N/A	
-----	--

maxSupply	int64	IWA Compatibility. Depends on TokenSupplyType. For tokens of type FUNGIBLECOMMON - the maximum number of tokens that can be in circulation. For tokens of type NONFUNGIBLEUNIQUE - the maximum number of NFTs (serial numbers) that can be minted. This field can never be changed
-----------	-------	--

feeScheduleKey	Key	The key which can change the token's custom fee schedule; must sign a TokenFeeScheduleUpdate transaction
----------------	-----	--

N/A	
-----	--

customFees	repeated CustomFee
The custom fees to be assessed during a CryptoTransfer that transfers units of this token	
N/A	
pausekey	Key
The Key which can pause and unpause the Token. If Empty the token pause status defaults to PauseNotApplicable, otherwise Unpaused	
N/A	

tokendelete.md:

TokenDelete

Marks a token as deleted, though it will remain in the ledger.

The operation must be signed by the specified Admin Key of the Token. If admin key is not set, Transaction will result in TOKENISIMMUTABLE.

Once deleted update, mint, burn, wipe, freeze, unfreeze, grant kyc, revoke kyc and token transfer transactions will resolve to TOKENWASDELETED.

TokenDeleteTransactionBody

Field	Type	Description
token	TokenID	The token to be deleted. If invalid token is specified, transaction will result in INVALIDTOKENID

tokendissociate.md:

TokenDissociate

Dissociates the provided account with the provided tokens. Must be signed by the provided Account's key.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALIDTOKENREF.

If any of the provided tokens has been deleted, the transaction will resolve to TOKENWASDELETED.

If an association between the provided account and any of the tokens does not exist, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If the provided account has a nonzero balance with any of the provided tokens, the transaction will resolve to TRANSACTIONREQUIRESZEROTOKENBALANCES.

TokenDissociateTransactionBody

Field	Type	Description

account	AccountID	The account to be dissociated with the provided tokens
tokens	repeated TokenID	The tokens to be dissociated with the provided account

tokenfeescheduleupdate.md:

TokenFeeScheduleUpdate

At consensus, updates a token type's fee schedule to the given list of custom fees. If the target token type has no fee\schedule\key, resolves to TOKENHASNOFEESCHEDULEKEY. \\\ Otherwise, this transaction must be signed to the fee\schedule\key, or the transaction will resolve to INVALIDSIGNATURE. If the custom\fees list is empty, clears the fee schedule or resolves to CUSTOMSCHEDULEALREADYHASNOFEES if the fee schedule was already empty.

Field	Type	Description
-----	-----	-----
-----	-----	-----
tokenId	TokenID	The token whose fee schedule is to be updated
customFees	repeated CustomFee	The new custom fees to be assessed during a CryptoTransfer that transfers units of this token

tokenfreezeaccount.md:

TokenFreezeAccount

Freezes transfers of the specified token for the account. Must be signed by the Token's freezeKey.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Frozen and will not be able to receive or send tokens unless unfrozen. The operation is idempotent.

TokenFreezeAccountTransactionBody

Field	Type	Description
-----	-----	-----
-----	-----	-----
token	TokenID	The token for which this account will be frozen. If


```
token does not exist, transaction results in INVALID\TOKEN\ID |
| account | AccountID | The account to be frozen
|
```

tokengetaccountnftinfo.md:

TokenGetAccountNftInfo

```
{% hint style="warning" %}
This query is currently deprecated.
{% endhint %}
```

TokenGetAccountNftInfoQuery

Applicable only to tokens of type NONFUNGIBLEUNIQUE. Gets info on NFTs N through M owned by the specified accountId. Example: If Account A owns 5 NFTs (might be of different Token Entity), having start=0 and end=5 will return all of the NFTs

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither)
accountId	AccountID	The Account for which information is requested
start	int64	Specifies the start index (inclusive) of the range of NFTs to query for. Value must be in the range \[0; ownedNFTs-1]
end	int64	Specifies the end index (exclusive) of the range of NFTs to query for. Value must be in the range (start; ownedNFTs]

TokenGetAccountNftInfoResponse

Response when the client sends the node TokenGetInfoQuery

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
nfts	repeated TokenNftInfo	List of NFTs associated to the account

tokengetinfo.md:

TokenGetInfo

TokenGetInfoQuery

Gets information about Token instance

Field	Type
Description	

```

-----
----- |
-----
----- |
| header | QueryHeader | Standard info sent from client to node, including the
signed payment, and what kind of response is requested (cost, state proof, both,
or neither) |
| token | TokenID
| The token for which information is requested. If invalid token is provided,
INVALID\TOKEN\ID response is returned. |

```

TokenGetInfoResponse

Response when the client sends the node TokenGetInfoQuery

```

| Field      | Type
| Description
| ----- |
-----
----- |
-----
| header      | ResponseHeader | Standard response from node to client, including
the requested fields: cost, or state proof, or both, or neither |
| tokenInfo | TokenInfo
| The information requested about this token instance
|

```

TokenInfo

The metadata about a Token instance

```

| Field      | Type
| Description
| ----- |
-----
----- |
-----
-----
-----
----- |
-----
| tokenId      | TokenID
| ID of the token instance
|
| name          | string
| The name of the token. It is a string of ASCII only characters
|
| symbol        | string
| The symbol of the token. It is a UTF-8 capitalized alphabetical string
|
| decimals      | uint32
| The number of decimal places a token is divisible by
|
| totalSupply   | uint64
| The total supply of tokens that are currently in circulation
|
| treasury      | AccountID      | The ID of the account which is set as
Treasury
|
| adminKey      | Key             | The key which can perform
update/delete operations on the token. If empty, the token can be perceived as

```

immutable (not being able to be updated/deleted)

| kycKey | Key | The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.

| freezeKey | Key | The key which can freeze or unfreeze an account for token transactions. If empty, freezing is not possible

| wipeKey | Key | The key which can wipe token balance of an account. If empty, wipe is not possible

| supplyKey | Key | The key which can change the supply of a token. The key is used to sign Token Mint/Burn operations

| defaultFreezeStatus | TokenFreezeStatus
| The default Freeze status (not applicable, frozen or unfrozen) of Hedera accounts relative to this token. FreezeNotApplicable is returned if Token Freeze Key is empty. Frozen is returned if Token Freeze Key is set and defaultFreeze is set to true. Unfrozen is returned if Token Freeze Key is set and defaultFreeze is set to false |

| defaultKycStatus | TokenKycStatus
| The default KYC status (KycNotApplicable or Revoked) of Hedera accounts relative to this token. KycNotApplicable is returned if KYC key is not set, otherwise Revoked

| isDeleted | bool
| Specifies whether the token was deleted or not

| autoRenewAccount | AccountID
| An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval

| autoRenewPeriod | uint64
| The interval at which the auto-renew account will be charged to extend the token's expiry

| expiry | uint64
| The epoch second at which the token will expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod

| memo | string
| The memo associated with the token

| tokenType | TokenType
| The token type

| supplyType | TokenSupplyType | The token supply type

| maxSupply | int64
| For tokens of type FUNGIBLECOMMON - The Maximum number of fungible tokens that can be in circulation. For tokens of type NONFUNGIBLEUNIQUE - the maximum number of NFTs (serial numbers) that can be in circulation

| feeschedulekey | Key
| The key which can change the custom fee schedule of the token; if not set, the fee schedule is immutable

| customfees | repeated CustomFee
| The custom fees to be assessed during a CryptoTransfer that transfers units of this token

| pausekey | Key

	The Key which can pause and unpaue the Token	
	pausestatus	TokenPauseStatus
	Specifies whether the token is paused or not. PauseNotApplicable is returned if pauseKey is not set.	
	ledgerid	bytes
	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs.	

###

tokengetnftinfo.md:

TokenGetNftInfo

NftID

Represents an NFT on the Ledger.

Field	Type	Description
-----	-----	
tokenID	TokenID	The (non-fungible) token of which this NFT is an instance
serialNumber	int64	The unique identifier of this instance

TokenGetNftInfoQuery

Applicable only to tokens of type NONFUNGIBLEUNIQUE. Gets info on a NFT for a given TokenID (of type NONFUNGIBLEUNIQUE) and serial number.

Field	Type	Description
-----	-----	
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither)
nftID	NftID	The ID of the NFT

TokenNftInfo

Field	Type	Description
-----	-----	
nftID	NftID	The ID of the NFT
accountID	AccountID	The current owner of the NFT
creationTime	Timestamp	The effective consensus timestamp at which the NFT was minted
metadata	bytes	Represents the unique metadata of the NFT

ledgerid	bytes	The ledger ID the response was returned from; please see HIP-198 for the network-specific IDs
spenderid	AccountID	If an allowance is granted for the NFT, its corresponding spender account

TokenGetNftInfoResponse

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
nft	TokenNftInfo	The information about this NFT

tokengetnftinfos.md:

TokenGetNftInfos

```
{% hint style="warning" %}
This query is deprecated.
{% endhint %}
```

TokenGetNftInfosQuery

Applicable only to tokens of type NONFUNGIBLEUNIQUE. Gets info on NFTs N through M on the list of NFTs associated with a given NONFUNGIBLEUNIQUE Token. Example: If there are 10 NFTs issued, having start=0 and end=5 will query for the first 5 NFTs. Querying +all 10 NFTs will require start=0 and end=10.

Field	Type	Description
header	QueryHeader	Standard info sent from client to node, including the signed payment, and what kind of response is requested (cost, state proof, both, or neither)
tokenId	TokenID	The ID of the token for which information is requested
start	int64	Specifies the start index (inclusive) of the range of NFTs to query for. Value must be in the range \[0; ownedNFTs-1]
end	int64	Specifies the end index (exclusive) of the range of NFTs to query for. Value must be in the range (start; ownedNFTs]

TokenGetNftInfosResponse

Field	Type	Description
header	ResponseHeader	Standard response from node to client, including the requested fields: cost, or state proof, or both, or neither
tokenId	TokenID	The Token with type NONFUNGIBLE that this record is for

| nfts | repeated TokenNftInfo | List of NFTs associated to the specified token |

tokengrانتkyc.md:

TokenGrantKyc

Grants KYC to the account for the given token. Must be signed by the Token's kycKey.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKEN\HAS\NO\KYC\KEY.

Once executed the Account is marked as KYC Granted.

TokenGrantKycTransactionBody

Field	Type	Description
token	TokenID	The token for which this account will be granted KYC. If token does not exist, transaction results in INVALID\TOKEN\ID
account	AccountID	The account to be KYCed

tokenmint.md:

TokenMint

Mints tokens to the Token's treasury Account. If no Supply Key is defined, the transaction will resolve to TOKENHASNOSUPPLYKEY.

The operation increases the Total Supply of the Token. The maximum total supply a token can have is $2^{63}-1$.

The amount provided must be in the lowest denomination possible.\nExample:Token A has 2 decimals. In order to mint 100 tokens, one must provide amount of 10000. In order to mint 100.55 tokens, one must provide amount of 10055.

TokenMintTransactionBody

Field	Type	Description
-------	------	-------------

```

-----
-----
-----
| token      | TokenID | The token for which to mint tokens. If token does not
exist, transaction results in INVALID\TOKEN\ID
|
| amount     | uint64          | Applicable to tokens of type
FUNGIBLECOMMON. The amount to mint to the Treasury Account. Amount must be a
positive non-zero number represented in the lowest denomination of the token.
The new supply must be lower than 2^63.
|
| metadata   | repeated bytes  | Applicable to tokens of type
NONFUNGIBLEUNIQUE. A list of metadata that are being created. Maximum allowed
size of each metadata is 100 bytes. The metadata can be arbitrary, but we
recommend developers to follow HIP-10standard. |

```

tokenpause.md:

TokenPause

Pauses the Token from being involved in any kind of Transaction until it is unpaused.

Must be signed with the Token's pause key.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If no Pause Key is defined, the transaction will resolve to TOKEN\HAS\NO\PAUSE\KEY.

Once executed the Token is marked as paused and will be not able to be a part of any transaction.

TokenPauseTransactionBody

Field	Type	Description
token	TokenID	The token to be paused

tokenrevokekyc.md:

TokenRevokeKyc

Revokes KYC to the account for the given token. Must be signed by the Token's kycKey.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKEN\HAS\NO\KYC\KEY.

Once executed the Account is marked as KYC Revoked

TokenRevokeKycTransactionBody

Field	Type	Description
-----	-----	-----
token	TokenID	The token for which this account will get his KYC revoked. If token does not exist, transaction results in INVALID\TOKEN\ID
account	AccountID	The account to be KYC Revoked

tokenservice.md:

TokenService

Transactions and queries for the Token Service.

{% hint style="info" %}
To transfer tokens created by HTS, please reference the cryptoTransfer API.
{% endhint %}

TokenService

Method Name	Request Type	Description
Response Type		
-----		-----
createToken	Transaction	Creates a new Token by submitting the transaction
TransactionResponse		
updateToken	Transaction	Updates the account by submitting the transaction
TransactionResponse		
mintToken	Transaction	Mints an amount of the token to the defined treasury account
TransactionResponse		
burnToken	Transaction	Burns an amount of the token from the defined treasury account
TransactionResponse		
deleteToken	Transaction	(NOT CURRENTLY SUPPORTED) Deletes a Token
TransactionResponse		
wipeTokenAccount	Transaction	Wipes the provided amount of tokens from the specified Account ID
TransactionResponse		
freezeTokenAccount	Transaction	Freezes the transfer of tokens to or from the specified Account ID
TransactionResponse		
unfreezeTokenAccount	Transaction	Unfreezes the transfer of tokens to or from the specified Account ID
TransactionResponse		
grantKycToTokenAccount	Transaction	Flags the provided Account ID as having gone through KYC
TransactionResponse		

revokeKycFromTokenAccount	Transaction	
TransactionResponse	Removes the KYC flag of the provided Account ID	
associateTokens	Transaction	
TransactionResponse	Associates tokens to an account	
dissociateTokens	Transaction	
TransactionResponse	Dissociates tokens from an account	
updateTokenFeeSchedule	Transaction	
TransactionResponse	Updates the custom fee schedule on a token	
getTokenInfo	Query	
Response	Retrieves the metadata of a token	
getAccountNftInfo	Query	
Response	Gets info on NFTs N through M on the list of	
NFTs associated with a given account		
getTokenNftInfo	Query	
Response	Retrieves the metadata of an NFT by TokenID	
and serial number		
getTokenNftInfo	Query	
Response	Gets info on NFTs N through M on the list of	
NFTs associated with a given Token of type NONFUNGIBLE		
pauseToken	Transaction	
TransactionResponse	Pause the token	
unpauseToken	Transaction	
TransactionResponse	Unpause the token	

tokenunfreezeaccount.md:

TokenUnfreezeAccount

Unfreezes transfers of the specified token for the account. Must be signed by the Token's freezeKey.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Unfrozen and will be able to receive or send tokens. The operation is idempotent.

TokenUnfreezeAccountTransactionBody

Field	Type	Description

Field	Type	Description
token	TokenID	The token for which this account will be unfrozen. If token does not exist, transaction results in INVALID\TOKEN\ID
account	AccountID	The account to be unfrozen

tokenunpause.md:

TokenUnpause

Unpauses the Token. Must be signed with the Token's pause key.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED

If no Pause Key is defined, the transaction will resolve to TOKEN\HAS\NO\PAUSE\KEY

Once executed the Token is marked as Unpaused and can be used in Transactions

The operation is idempotent - becomes a no-op if the Token is already unpaused

TokenUnpauseTransactionBody

Field	Type	Description
token	TokenID	The token to be unpaused

tokenupdate.md:

TokenUpdate

Updates an already created Token.

If no value is given for a field, that field is left unchanged. For an immutable token (that is, a token created without an adminKey), only the expiry may be updated. Setting any other field, in that case, will cause the transaction status to resolve to TOKEN\IS\IMMUTABLE.

TokenUpdateTransactionBody

Field	Type	Description
Signature Required		
token	TokenID	The Token to be updated
N/A		
symbol	string	The new publicly visible symbol of the token. The token name is specified as a Unicode string. Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0 byte (NUL).
N/A		
name	string	The new publicly visible name of the token. The token name is specified as a Unicode string. Its UTF-8 encoding cannot exceed 100 bytes, and cannot contain the 0 byte (NUL).
N/A		
treasury	AccountID	The new Treasury account of the Token. If the provided treasury account is not existing or deleted, the response will be INVALID\TREASURY\ACCOUNT\FOR\TOKEN. If successful, the Token balance held in the

previous Treasury Account is transferred to the new one. | If updated, required

adminKey	Key	The new Admin key of the Token. If Token
is immutable, transaction will resolve to TOKEN\IS\IMMUTABLE.		
If updated, required		
kycKey	Key	The new KYC key of the Token. If Token
does not have currently a KYC key, transaction will resolve to TOKEN\HAS\NO\KYC\KEY.		
If updated, required		
freezeKey	Key	The new Freeze key of the Token. If the
Token does not have currently a Freeze key, transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.		
If updated, required		
wipeKey	Key	The new Wipe key of the Token. If the
Token does not have currently a Wipe key, transaction will resolve to TOKEN\HAS\NO\WIPE\KEY.		
If updated, required		
supplyKey	Key	The new Supply key of the Token. If the
Token does not have currently a Supply key, transaction will resolve to TOKEN\HAS\NO\SUPPLY\KEY.		
If updated, required		
autoRenewAccount	AccountID	The new account which will be automatically
charged to renew the token's expiration, at autoRenewPeriod interval.		
N/A		
autoRenewPeriod	uint64	The new interval
at which the auto-renew account will be charged to extend the token's expiry.		
N/A		
expiry	uint64	The new expiry
time of the token. Expiry can be updated even if admin key is not set. If the		
provided expiry is earlier than the current token expiry, transaction will		
resolve to INVALID\EXPIRATION\TIME		
N/A		
memo	string	The memo
associated with the token (UTF-8 encoding max 100 bytes)		
N/A		
feeScheduleKey	Key	If set, the new key to use to update the
token's custom fee schedule; if the token does not currently have this key,		
transaction will resolve to TOKEN\HAS\NO\FEE\SCHEDULE\KEY		
N/A		
pausekey	Key	The Key which can
pause and unpause the Token. If the Token does not currently have a pause key,		
transaction will resolve to TOKEN\HAS\NO\PAUSE\KEY		
N/A		

tokenwipeaccount.md:

TokenWipeAccount

Wipes the provided amount of tokens from the specified Account. Must be signed by the Token's Wipe key.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKENWASDELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If Wipe Key is not present in the Token, the transaction results in TOKENHASNOWIPEKEY.

If the provided account is the Token's Treasury Account, the transaction results in CANNOTWIPETOKENTREASURYACCOUNT

On success, tokens are removed from the account and the total supply of the token is decreased by the wiped amount.

The amount provided is in the lowest denomination possible. Example:

Token A has 2 decimals. In order to wipe 100 tokens from account, one must provide amount of 10000. In order to wipe 100.55 tokens, one must provide amount of 10055.

TokenWipeAccountTransactionBody

Field	Type	Description
token	TokenID	The token for which the account will be wiped. If token does not exist, transaction results in INVALIDTOKENID
account	AccountID	The account to be wiped
amount	uint64	Applicable to tokens of type FUNGIBLECOMMON. The amount of tokens to wipe from the specified account. Amount must be a positive non-zero number in the lowest denomination possible, not bigger than the token balance of the account (0; balance]
serialNumbers	int64	Applicable to tokens of type NONFUNGIBLEUNIQUE. The list of serial numbers to be wiped.

assessedcustomfee.md:

AssessedCustomFee

A custom transfer fee that was assessed during the handling of a CryptoTransfer.

Field	Type	Description
amount	int64	The number of units assessed for the fee
tokenid	TokenID	The denomination of the fee; taken as hbar if left unset
feecollectoraccountid	AccountID	The account to receive the assessed fee
effectivepayeraccountid	repeated AccountID	The account(s) whose final balances would have been higher in the absence of this assessed fee

customfee.md:

CustomFee

A transfer fee to assess during a CryptoTransfer that transfers units of the token to which the fee is attached. A custom fee may be either fixed or fractional, and must specify a fee collector account to receive the assessed fees. Only positive fees may be assessed.

Field	Type	
Description		
-----	-----	
one of fee {		
fixedfee	FixedFee	Fixed fee to be charged
fractionalfree	FractionalFee	Fractional fee to be charged
royaltyfee	RoyaltyFee	Royalty fee to be charged
}		
feecollectoraccountid	AccountID	The account to receive the custom fee

fixedfee.md:

FixedFee

A fixed number of units (hbar or token) to assess as a fee during a CryptoTransfer that transfers units of the token to which this fixed fee is attached.

Field	Type	
Description		
-----	-----	
amount	int64	The number of units to assess as a fee
denominatingtokenid	TokenID	The denomination of the fee; taken as hbar if left unset

fractionalfee.md:

FractionalFee

A fraction of the transferred units of a token to assess as a fee. The amount assessed will never be less than the given minimum\amount, and never greater than the given maximum\amount. The denomination is always units of the token to which this fractional fee is attached.

Field	Type	Description
-----	-----	
fractionalamount	Fraction	The fraction of the transferred units to assess as a fee

minimumamount	int64	The minimum amount to assess
maximumamount	int64	The maximum amount to assess (zero implies no maximum)
netoftransfers	bool	If true, assesses the fee to the sender, so the receiver gets the full amount from the token transfer list, and the sender is charged an additional fee; if false, the receiver does NOT get the full amount, but only what is left over after paying the fractional fee

README.md:

CustomFees

royaltyfee.md:

RoyaltyFee

A fee to assess during a CryptoTransfer that changes ownership of an NFT. Defines the fraction of the fungible value exchanged for an NFT that the ledger should collect as a royalty. ("Fungible value" includes both \hbar and units of fungible HTS tokens.) When the NFT sender does not receive any fungible value, the ledger will assess the fallback fee, if present, to the new NFT owner. Royalty fees can only be added to tokens of type NONFUNGIBLEUNIQUE.

Field	Type	Description
exchangevaluefraction	Fraction	The fraction of fungible value exchanged for an NFT to collect as royalty
fallbackfee	FixedFee	If present, the fixed fee to assess to the NFT receiver when no fungible value is exchanged with the sender

client.md:

Build Your Hedera Client

1. Configure your Hedera Network

Build your client to interact with any of the Hedera network nodes. Mainnet, testnet, and previewnet are the three Hedera networks you can submit transactions and queries to.

For a predefined network (preview, testnet, and mainnet), the mirror node client is configured to the corresponding network mirror node. The default mainnet mirror node connection is to the whitelisted mirror node.

To access the public mainnet mirror node, use setMirrorNetwork() and enter mainnet-public.mirrornode.hedera.com:433 for the endpoint. The gRPC API requires TLS. The following SDK versions are compatible with TLS:

Java: v2.3.0+
 JavaScript: v2.4.0+
 Go: v2.4.0+

<table data-header-hidden><thead><tr><th width="332"></th><th>

width="211.33333333333331"></th><th></th></tr></thead><tbody><tr><td>Method</td><td>Type</td><td>Description</td></tr><tr><td><code>Client.forPreviewnet()</code></td><td></td><td><code>Constructs a Hedera client pre-configured for Previewnet access</code></td></tr><tr><td><code>Client.forTestnet()</code></td><td></td><td><code>Constructs a Hedera client pre-configured for Testnet access</code></td></tr><tr><td><code>Client.forMainnet()</code></td><td></td><td><code>Constructs a Hedera client pre-configured for Mainnet access</code></td></tr><tr><td><code>Client.forNetwork(<network>)</code></td><td><code>Map<String, AccountId></code></td><td><code>Construct a client given a set of nodes. It is the responsibility of the caller to ensure that all nodes in the map are part of the same Hedera network. Failure to do so will result in undefined behavior.</code></td></tr><tr><td><code>Client.fromConfig(<json>)</code></td><td><code>String</code></td><td><code>Configure a client from the given JSON string describing a ClientConfiguration object</code></td></tr><tr><td><code>Client.fromConfig(<json>)</code></td><td><code>Reader</code></td><td><code>Configure a client from the given JSON reader</code></td></tr><tr><td><code>Client.fromConfigFile(<file>)</code></td><td><code>File</code></td><td><code>Configure a client based on a JSON file.</code></td></tr><tr><td><code>Client.fromConfigFile(<fileName>)</code></td><td><code>String</code></td><td><code>Configure a client based on a JSON file at the given path.</code></td></tr><tr><td><code>Client.forName(<name>)</code></td><td><code>String</code></td><td><code>Provide the name of the network. <code>mainnet</code> <code>testnet</code> <code>previewnet</code></code></td></tr><tr><td><code>Client.<network>.setMirrorNetwork(<network>)</code></td><td><code>List<String></code></td><td><code>Define a specific mirror network node(s) ip:port in string format</code></td></tr><tr><td><code>Client.<network>.getMirrorNetwork()</code></td><td><code>List<String></code></td><td><code>Return the mirror network node(s) ip:port in string format</code></td></tr><tr><td><code>Client.setTransportSecurity()</code></td><td><code>boolean</code></td><td><code>Set if transport security should be used. If transport security is enabled all connections to nodes will use TLS, and the server's certificate hash will be compared to the hash stored in the node address book for the given network.</code></td></tr><tr><td><code>Client.setNetworkUpdatePeriod()</code></td><td><code>Duration</code></td><td><code>Client automatically updates the network via a mirror node query at regular intervals. You can set the interval at which the address book is updated.</code></td></tr><tr><td><code>Client.setNetworkFromAddressBook(<addressBook>)</code></td><td><code>AddressBook</code></td><td><code>Client can be set from a <code>NodeAddressBook</code>.</code></td></tr><tr><td><code>Client.setLedgerId(<ledgerId>)</code></td><td><code>LedgerId</code></td><td><code>The ID of the network. <code>LedgerId.MAINNET</code> <code>LedgerId.TESTNET</code> <code>LedgerId.PREVIEWNET</code></code></td></tr><tr><td><code>Client.getLedgerId()</code></td><td><code>LedgerId</code></td><td><code>Get the ledger ID</code></td></tr><tr><td><code>Client.setVerifyCertificates()</code></td><td><code>boolean</code></td><td><code>Set if server certificates should be verified against an existing address book.</code></td></tr></tbody></table>

```
{% tabs %}
{% tab title="Java" %}
java
// From a pre-configured network
Client client = Client.forTestnet();

//For a specified network
Map<String, AccountId> nodes = new HashMap<>();
nodes.put("34.94.106.61:50211" ,AccountId.fromString("0.0.10"));
```

```

Client.forNetwork(nodes);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
// From a pre-configured network
const client = Client.forTestnet();

//For a specified network
const nodes = {"34.94.106.61:50211": new AccountId(10)}
const client = Client.forNetwork(nodes);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
// From a pre-configured network
client := hedera.ClientForTestnet()

//For a specified network
node := map[string]AccountId{
    "34.94.106.61:50211": {Account: 10}
}

client := Client.forNetwork(nodes)

//v2.0.0

{% endtab %}
{% endtabs %}

```

2. Define the operator account ID and private key

The operator is the account that will, by default, pay the transaction fee for transactions and queries built with this client. The operator account ID is used to generate the default transaction ID for all transactions executed with this client. The operator private key is used to sign all transactions executed by this client.

Method	Type
Client.<network>.setOperator(<accountId, privateKey>)	AccountId, PrivateKey
Client.<network>.setOperatorWith(<accountId, privateKey, transactionSigner>)	AccountId, PrivateKey, Function\<byte\[, byte \[,]>

From an account ID and private key

```

{% tabs %}
{% tab title="Java" %}
java
// Operator account ID and private key from string value
AccountId MYACCOUNTID = AccountId.fromString("0.0.96928");
Ed25519PrivateKey MYPRIVATEKEY =
PrivateKey.fromString("302e020100300506032b657004220420b9c3ebac81a72aafa5490cc78
111643d016d311e60869436fbb91c7330796928");

```



```

// Pre-configured client for test network (testnet)
Client client = Client.forTestnet()

//Set the operator with the account ID and private key
client.setOperator(MYACCOUNTID, MYPRIVATEKEY);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Your account ID and private key from string value
const MYACCOUNTID = AccountId.fromString("0.0.96928");
const MYPRIVATEKEY =
PrivateKey.fromString("302e020100300506032b657004220420b9c3ebac81a72aafa5490cc78
111643d016d311e60869436fbb91c7330796928");

// Pre-configured client for test network (testnet)
const client = Client.forTestnet()

//Set the operator with the account ID and private key
client.setOperator(MYACCOUNTID, MYPRIVATEKEY);

{% endtab %}

{% tab title="Go" %}
go
// Operator account ID and private key from string value
operatorAccountID, err := hedera.AccountIDFromString("0.0.96928")
if err != nil {
    panic(err)
}

operatorKey, err :=
hedera.PrivateKeyFromString("302e020100300506032b65700422042012a4a4add3d885bd61d
7ce5cff88c5ef2d510651add00a7f64cb90de33596928")
if err != nil {
    panic(err)
}

// Pre-configured client for test network (testnet)
client := hedera.ClientForTestnet()

//Set the operator with the operator ID and operator key
client.SetOperator(operatorAccountID, operatorKey)

{% endtab %}
{% endtabs %}

```

From a .env file

The .env file is created in the root directory of the SDK. The .env file stores account ID and the associated private key information to reference throughout your code. You will need to import the relevant dotenv module to your project files. The sample .env file may look something like this:

.env

```

MYACCOUNTID=0.0.941
MYPRIVATEKEY=302e020100300506032b65700422042012a4a4add3d885bd61d7ce5cff88c5ef2d5
10651add00a7f64cb90de3359bc5e

```

```

{% tabs %}

```

```

{% tab title="Java" %}
java
//Grab the account ID and private key of the operator account from the .env file
AccountId MYACCOUNTID =
AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATORID")));
Ed25519PrivateKey MYPRIVATEKEY =
Ed25519PrivateKey.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATORKEY")));

// Pre-configured client for test network (testnet)
Client client = Client.forTestnet()

//Set the operator with the account ID and private key
client.setOperator(MYACCOUNTID, MYPRIVATEKEY);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Grab the account ID and private key of the operator account from the .env file
const myAccountId = process.env.MYACCOUNTID;
const myPrivateKey = process.env.MYPRIVATEKEY;

// Pre-configured client for test network (testnet)
const client = Client.forTestnet()

//Set the operator with the account ID and private key
client.setOperator(myAccountId, myPrivateKey);

{% endtab %}

{% tab title="Go" %}
go
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from demo.env
file. Error:\n%\v\n", err))
    }

    //Get the operator account ID and private key
    MYACCOUNTID := os.Getenv("MYACCOUNTID")
    MYPRIVATEKEY := os.Getenv("MYPRIVATEKEY")

    myAccountId, err := hedera.AccountIDFromString(MYACCOUNTID)
    if err != nil {
        panic(err)
    }

    myPrivateKey, err := hedera.PrivateKeyFromString(MYPRIVATEKEY)
    if err != nil {
        panic(err)
    }

{% endtab %}
{% endtabs %}

```

3. Additional client modifications

```

{% hint style="warning" %}
The max transaction fee and max query payment are both set to 100\000\000
tinybar (1 HBAR). This amount can be modified by using
setDefaultMaxTransactionFee()and setDefaultMaxQueryPayment().
{% endhint %}

```

Method	Type	Description
<code>Client.<network>.setDefaultRegenerateTransactionId(<regenerateTransactionId>)</code>	boolean	Whether or not to regenerate the transaction IDs
<code>Client.<network>.getDefaultRegenerateTransactionId(<regenerateTransactionId>)</code>	boolean	Get the default regenerate transaction ID
<code>Client.<network>.setDefaultMaxTransactionFee(<fee>)</code>	Hbar	The maximum transaction fee the client is willing to pay
<code>Client.<network>.getDefaultMaxTransactionFee()</code>	Hbar	Get the default max transaction fee that is set
<code>Client.<network>.setDefaultMaxQueryPayment(<maxQueryPayment>)</code>	Hbar	The maximum query payment the client will pay.
<code>Client.<network>.getDefaultMaxQueryPayment()</code>	Hbar	Get the default max query payment
<code>Client.<network>.setNetwork(<nodes>)</code>	Map<String, AccountId>	Replace all nodes in this Client with a new set of nodes (e.g. for an Address Book update)
<code>Client.<network>.getNetwork()</code>	Map<String, AccountId>	Get the network nodes
<code>Client.<network>.setRequestTimeout(<requestTimeout>)</code>	Duration	The period of time a transaction or query request will retry from a "busy" network response
<code>Client.<network>.getRequestTimeout()</code>	Duration	Get the period of time a transaction or query request will retry from a "busy" network response
<code>Client.<network>.setMinBackoff(<minBackoff>)</code>	Duration	The minimum amount of time to wait between retries. When retrying, the delay will start at this time and increase exponentially until it reaches the
<code>Client.<network>.getMinBackoff()</code>	Duration	Get the minimum amount of time to wait between retries
<code>Client.<network>.setMaxBackoff(<maxBackoff>)</code>	Duration	The maximum amount of time to wait between retries. Every retry attempt will increase the wait time exponentially until it reaches this
<code>Client.<network>.getMaxBackoff()</code>	Duration	Get the maximum amount of time to wait between retries
<code>Client.<network>.setAutoValidateChecksums(<value>)</code>	boolean	Validate checksums
<code>Client.<network>.setCloseTimeout(<closeTimeout>)</code>	Duration	Timeout for closing either a single node when setting a new network, or closing the entire network
<code>Client.<network>.setMaxNodeAttempts(<maxNodeAttempts>)</code>	int	Set the max number of times a node can return a bad gRPC status before we remove it from the list
<code>Client.<network>.getMaxNodeAttempts()</code>	int	Get the max node attempts
<code>Client.<network>.setMinNodeReadmitTime(<minNodeReadmitTime>)</code>	Duration	The min time to wait before attempting to readmit nodes
<code>Client.<network>.getMinNodeReadmitTime()</code>	Duration	Get the minimum node readmit time
<code>Client.<network>.setMaxNodeReadmitTime(<maxNodeReadmitTime>)</code>	Duration	The max time to wait before attempting to readmit nodes
<code>Client.<network>.getMaxNodeReadmitTime()</code>	Duration	Get the max node readmit time

{% tabs %}

```

{% tab title="Java" %}
java
// For test network (testnet)
Client client = Client.forTestnet()

//Set your account as the client's operator
client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(10));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(5));

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
{% code title="JavaScript" %}
java
// For test network (testnet)
const client = Client.forTestnet()

//Set your account as the client's operator
client.setOperator(myAccountId, myPrivateKey);

//Set the default maximum transaction fee (in Hbar)
client.setDefaultMaxTransactionFee(new Hbar(10));

//Set the maximum payment for queries (in Hbar)
client.setDefaultMaxQueryPayment(new Hbar(5));

//v2.0.0

{% endcode %}
{% endtab %}

{% tab title="Go" %}
go
// For test network (testnet)
client := hedera.ClientForTestnet()

//Set your account as the client's operator
client.SetOperator(myAccountId, myPrivateKey)

// Set default max transaction fee
client.SetDefaultMaxTransactionFee(hedera.HbarFrom(10, hedera.HbarUnits.Hbar))

// Set max query payment
client.setDefaultMaxQueryPayment(hedera.HbarFrom(5, hedera.HbarUnits.Hbar))

//v2.0.0

{% endtab %}
{% endtabs %}

```

general-errors.md:

General Network Response Messages

General network response messages and their descriptions.

Error	Description
-----	-----
AUTORENEWDURATIONNOTINRANGE	The duration is not a subset of \[MINIMUM\AUTORENEW\DURATION, MAXIMUM\AUTORENEW\DURATION]
ACCOUNTISNOTGENESISACCOUNT	Special Account Operations should be performed by only Genesis account, return this code if it is not Genesis Account
DUPLICATETRANSACTION	This transaction ID is a duplicate of one that was submitted to this node or reached consensus in the last 180 seconds (receipt period)
ENTITYNOTALLOWEDTODELETE	Entities with Entity ID below 1000 are not allowed to be deleted
EMPTYQUERYBODY	The query body is empty
EMPTYTRANSACTIONBODY	Transaction body provided is empty
FAILINVALID	There was a system error and the transaction failed because of invalid request parameters.
FREEZETRANSACTIONBODYNOTFOUND	FreezeTransactionBody does not exist
INSUFFICIENTPAYERBALANCE	The payer account has insufficient cryptocurrency to pay the transaction fee
INSUFFICIENTTXFEE	The fee provided in the transaction is insufficient for this type of transaction
INVALIDFEESUBMITTED	Invalid fee submitted
INVALIDFREEZETRANSACTIONBODY	FreezeTransactionBody is invalid
INVALIDNODEACCOUNT	Node Account provided does not match the node account of the node the transaction was submitted to.
INVALIDPAYERSIGNATURE	Payer signature is invalid
INVALIDRECEIVINGNODEACCOUNT	In Query validation, an account with +ve(amount) value should be Receiving node account, the receiver account should be only one account in the list
INVALIDRENEWALPERIOD	Auto-renewal period is not a positive number of seconds
INVALIDTRANSACTION	For any error not handled by specific error codes listed below.
INVALIDTRANSACTIONBODY	Invalid transaction body provided
INVALIDTRANSACTIONID	The transaction id is not valid
INVALIDTRANSACTIONDURATION	Valid transaction duration is a positive non zero number that does not exceed 120 seconds
INVALIDTRANSACTIONSTART	Transaction start time is greater than the current consensus time
INVALIDSIGNATURE	The transaction signature is not valid
OK	The transaction passed the precheck validations.

MEMOTOOLONG	Transaction memo size exceeded 100 bytes
MISSINGQUERYHEADER	Header is missing in Query request
NOTSUPPORTED	The API is not currently supported
PAYERACCOUNTNOTFOUND	Payer account does not exist.
PAYERACCOUNTUNAUTHORIZED	The fee payer account doesn't have permission to submit such Transaction
PLATFORMNOTACTIVE	The platform node is either disconnected or lagging behind
PLATFORMTRANSACTIONNOTCREATED	Transaction not created by platform due to large backlog
RECEIPTNOTFOUND	Receipt for given transaction id does not exist
RECORDNOTFOUND	Record for given transaction id does not exist
SUCCESS	The transaction succeeded
TRANSACTIONEXPIRED	Pre-Check error when TransactionValidStart + transactionValidDuration is less than current consensus time.
TRANSACTIONOVERSIZE	The size of the Transaction is greater than transactionMaxBytes
TRANSACTIONTOOMANYLAYERS	The Transaction has more than 50 levels
UNKNOWN	This node has submitted this transaction to the network. Status of the transaction is currently unknown.

Keys

Error	Description

BADENCODING	Unsupported algorithm/encoding used for keys in the transaction
INVALIDKEYENCODING	Provided key encoding was not supported by the system
INVALIDSIGNATURECOUNTMISMATCHINGKEY	The number of key (KeyList, or ThresholdKey) does not match that of signature (SignatureList, or ThresholdKeySignature). e.g. if a keyList has 3 base keys, then the corresponding signatureList should also have 3 base signatures.
INVALIDSIGNATURETYPEPEMISMATCHING	The type of key (base ed25519 key, KeyList, or ThresholdKey) does not match the type of signature (base ed25519 signature, SignatureList, or ThresholdKeySignature)
KEYNOTPROVIDED	The keys were not provided in the request.
KEYPREFIXMISMATCH	One public key matches more than one

prefixes on the signature map

```
|
| KEYREQUIRED | Key not provided in the transaction
body
|
```

hbars.md:

HBAR

Constructor	Type	Description
new Hbar(<amount>)	Hbar	Initializes the Hbar object

```
java
new Hbar(<amount>)
```

HBAR from:

Construct HBAR from different representations.

Method	Type	Description
Hbar.from(<hbars>)	long / BigDecimal	Returns an Hbar whose value is equal to the specified value
Hbar.from(<hbars, unit>)	long / BigDecimal , HbarUnit	Returns an Hbar representing the value in the given units
Hbar.fromString(<text>)	CharSequence	Converts the provided string into an amount of hbars
Hbar.fromString(<text, unit>)	CharSequence, HbarUnit	Converts the provided string into an amount of hbars
Hbar.fromTinybars(<tinybars>)	long	Returns an Hbar converted from the specified number of tinybars

```
{% tabs %}
{% tab title="Java" %}
```

```
java
//10 HBAR
new Hbar(10);
```

```
//10 HBAR from hbar value
Hbar.from(10);
```

```
//100 tinybars from HBAR convert to unit
Hbar.from(100, HbarUnit.TINYBAR);
```

```
// 10 HBAR converted from string value
Hbar.fromString("10");
```

```
//100 tinybars from string value
Hbar.fromString("10", HbarUnit.TINYBAR);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
// 10 HBAR
new Hbar(10);
```

```
//10 HBAR
Hbar.from(10);

//100 tinybars
Hbar.from(100, HbarUnit.TINYBAR);

// 10 HBAR converted from string value
Hbar.fromString("10");

//100 tinybars from string value
Hbar.fromString("100", HbarUnit.TINYBAR);

{% endtab %}

{% tab title="Go" %}
go
//100 HBAR
hedera.NewHbar(10)

//100 tinybars
hedera.HbarFrom(10, hedera.HbarUnits.Tinybar)

//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

HBAR to:

Convert HBAR to a different unit/format.

Method	Type	Description
to(<unit>)	HbarUnit	Specify the unit of hbar to convert to. Use As for Go.
toString(<unit>)	HbarUnit	String value of the hbar unit to convert to. Use String() for Go.
toTinybars()	Long	Hbar value converted to tinybars

```
{% tabs %}
{% tab title="Java" %}
java
//10 HBAR converted to tinybars
new Hbar(10).to(HbarUnit.TINYBAR);

//10 HBAR converted to tinybars
new Hbar(10).toString(HbarUnit.TINYBAR);

//10 HBAR converted to tinybars
new Hbar(10).toTinybars();

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//10 HBAR converted to tinybars
new Hbar(10).to(HbarUnit.TINYBAR);
```



```
//10 HBAR converted to tinybars
new Hbar(10).toString(HbarUnit.TINYBAR);

//10 HBAR converted to tinybars
new Hbar(10).toTinybars();

{% endtab %}

{% tab title="Go" %}
go
//10 HBAR converted to tinybars
hedera.NewHbar(10).As(hedera.HbarUnits.Tinybar)

//10 HBAR to string format
hedera.NewHbar(10).String()

//10 HBAR converted to tinybars
hedera.NewHbar(10).AsTinybar()
//v2.0.0

{% endtab %}
{% endtabs %}
```

HBAR constants:

Provided constant values of HBAR.

Method	Type	Description
Hbar.MAX	Hbar	A constant value of the maximum number of hbars (50\000\000\000 hbars)
Hbar.MIN	Hbar	A constant value of the minimum number of hbars (-50\000\000\000 hbars)
Hbar.ZERO	Hbar	A constant value of zero hbars

```
{% tabs %}
{% tab title="Java" %}
java
//The maximum number of hbars
Hbar hbarMax = Hbar.MAX;

//The minimum number of hbars
Hbar hbarMin = Hbar.MIN;

//A constant value of zero hbars
Hbar hbarZero = Hbar.ZERO;

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//The maximum number of hbars
const hbarMax = Hbar.MAX;

//The minimum number of hbars
const hbarMin = Hbar.MIN;

//A constant value of zero hbars
const hbarZero = Hbar.ZERO;
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
//The maximum number of hbars
```

```
hbarMax := hedera.MaxHbar
```

```
//The minimum number of hbars
```

```
hbarMin := hedera.MinHbar
```

```
//A constant value of zero hbars
```

```
hbarZero := hedera.ZeroHbar
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

HBAR units

Modify the HBAR representation to one of the HBAR denominations.

Function	Description
----------	-------------

-------	--

--	--

--	--

HbarUnit.TINYBAR	The atomic (smallest) unit of hbar, used natively by the Hedera network
------------------	---

HbarUnit.MICROBAR	Equivalent to 100 tinybar or 1/1,000,000 hbar.
-------------------	--

--	--

HbarUnit.MILLIBAR	Equivalent to 100,000 tinybar or 1/1,000 hbar
-------------------	---

--	--

HbarUnit.HBAR	The base unit of hbar, equivalent to 100 million tinybar.
---------------	---

--	--

HbarUnit.KILOBAR	Equivalent to 1 thousand hbar or 100 billion
------------------	--

tinybar.HbarUnit.Megabar	
--------------------------	--

HbarUnit.MEGABAR	Equivalent to 1 million hbar or 100 trillion tinybar.
------------------	---

--	--

HbarUnit.GIGABAR	Equivalent to 1 billion hbar or 100 quadrillion tinybar.
------------------	--

--	--

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//100 tinybars
```

```
Hbar.from(100, HbarUnit.TINYBAR);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//100 tinybars
```

```
Hbar.from(100, HbarUnit.TINYBAR);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
//100 tinybars
```

```
hedera.HbarFrom(100, hedera.HbarUnits.Tinybar)
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

HBAR decimal places

The decimal precision of HBAR varies across the different Hedera APIs. While HAPI, JSON-RPC Relay, and Hedera Smart Contract Service (EVM) provide 8 decimal places, the `msg.value` in JSON-RPC Relay provides 18 decimal places.

width="495">API	Decimal
Hedera API (HAPI) (Crypto + SCS Service (<code>msg.value</code>))	8
Hedera Smart Contract Service (EVM)	8
JSON RPC Relay (passed as arguments)	8
JSON RPC Relay (<code>msg.value</code>)	18

```
{% hint style="warning" %}
```

Note: The JSON-RPC Relay `msg.value` uses 18 decimals when it returns HBAR. As a result, the `gasPrice` also uses 18 decimal places since it is only utilized from the JSON-RPC Relay.

```
{% endhint %}
```

pseudorandom-number-generator.md:

Pseudorandom Number Generator

A transaction that generates a pseudorandom number. When the pseudorandom number generate transaction executes, its transaction record will contain the 384-bit array of pseudorandom bytes. The transaction has an optional range parameter. If the parameter is given and is positive, then the record will contain a 32-bit pseudorandom integer r , where $0 \leq r < \text{range}$ instead of containing the 384 pseudorandom bits.

When the n th transaction needs a pseudorandom number, it is given the running hash of all records up to and including the record for transaction $n-3$. If it needs 384 bits, then it uses the entire hash. If it needs 256 bits, it uses the first 256 bits of the hash. If it needs a random number r that is in the range $0 \leq r < \text{range}$, then it lets x be the first 32 bits of the hash (interpreted as a signed integer).

The choice of using the hash up to transaction $n-3$ rather than $n-1$ is to ensure the transactions can be processed quickly. Because the thread calculating the hash will have more time to complete it before it is needed. The use of $n-3$ rather than $n-1000000$ is to make it hard to predict the pseudorandom number in advance.\

Reference: HIP-351

Field	Description
Range	The specified range to return the pseudorandom number from.

Methods

Method	Type	Requirement
<code>setRange(<range>)</code>	integer	Optional
<code>getRange(<range>)</code>	integer	Optional

```

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction with range set
TransactionResponse transaction = new PrngTransaction()
    //Set the range
    .setRange(250)
    .execute(client);

//Get the record
TransactionRecord transactionRecord = transaction.getRecord(client);

//Get the number
int prngNumber = transactionRecord.prngNumber;

System.out.println(prngNumber);

//SDK version 2.17.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction with range set
const transaction = await new PrngTransaction()
    //Set the range
    .setRange(250)
    .execute(client);

//Get the record
const transactionRecord = await transaction.getRecord(client);

//Get the number
const prngNumber = transactionRecord.prngNumber;

console.log(prngNumber);

//SDK version 2.17.0

{% endtab %}

{% tab title="Go" %}
go
transaction, err := hedera.NewPrngTransaction().
    // Set the range
    SetRange(250).
    Execute(client)
if err != nil {
    println(err.Error(), ": error executing rng transaction")
    return
}

transactionRecord, err := createResponse.GetRecord(client)
if err != nil {
    println(err.Error(), ": error getting receipt")
    return
}

if transactionRecord.PrngNumber != nil {
    println(err.Error(), ": error, pseudo-random number is nil")
    return
}

```

```
println("The pseudorandom number is:", transactionRecord.PrngNumber)
```

```
//Version 2.17.0
```

```
{% endtab %}  
{% endtabs %}
```

```
# queries.md:
```

Queries

Queries are requests that do not require network consensus. Queries are processed only by the single node the request is sent to. Below is a list of network queries by service.

Cryptocurrency Accounts	Consensus
Tokens	File Service
Smart Contracts	
Schedule Service	

AccountBalanceQuery TopicInfoQuery TokenBalanceQuery	
FileContentsQuery ContractCallQuery ScheduleInfoQuery	
AccountInfoQuery TopicMessageQuery TokenInfoQuery	
FileInfoQuery ContractByteCodeQuery	

Get Query Cost

A query that returns the cost of a query prior to submitting the query to the network node for processing. If the cost of the query is greater than the default max query payment (1 HBAR) you can use `setMaxQueryPayment(<hbar>)` to change the default.

Method	Type	Description
<code>getCost(&#x3C;client>)</code>	Client	Get the cost of the query in HBAR
<code>getCost(&#x3C;client, timeout>)</code>	Client, Duration	The max length of time the SDK will attempt to retry in the event of repeated busy responses from the node(s)
<code>getCostAsync(&#x3C;client>)</code>	Client	Get the cost of a query asynchronously

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Create the query request  
AccountBalanceQuery query = new AccountBalanceQuery()  
    .setAccountId(accountId);
```

```
//Get the cost of the query  
Hbar queryCost = query.getCost(client);
```

```
System.out.println("The account balance query cost is " +queryCost);
```

```
//v2.0.0
```

```
{% endtab %}
```

```

{% tab title="JavaScript" %}
javascript
//Create the query request
const query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Get the cost of the query
const queryCost = await query.getCost(client);

console.log("The account balance query cost is " +queryCost);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the query request
query := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Get the cost of the query
cost, err := query.GetCost(client)

if err != nil {
    panic(err)
}

fmt.Printf("The account balance query cost is: %v\n ", cost.String())

//v2.0.0

{% endtab %}
{% endtabs %}

# README.md:

---
description: Hedera supported and community-maintained SDKs
cover: ../../.gitbook/assets/Hero-Desktop-Tooling2022-12-07-021130ayix.webp
coverY: -154.63917525773195
---

SDKs

For friendly, language-specific access to the Hedera API and its network
services, there are several options.

Hedera Services Code SDKs

Hedera and the developer community contribute to and maintain Hedera Services
Code SDKs across various languages.

{% hint style="info" %}
Note: The Hedera JavaScript SDK supports React Native with Expo on Android
devices and Android emulators. It does not currently support React Native Bare.
{% endhint %}

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th align="center"></th><th align="center"></th><th
align="center"></th><th data-hidden data-card-cover data-type="files"></th><th
data-hidden data-card-target

```

Hedera Java SDK	Maintainer: Hedera	License: Apache 2.0	DOWNLOAD
https://github.com/hashgraph/hedera-sdk-java			
Hedera JavaScript SDK	Maintainer: Hedera	License: Apache 2.0	DOWNLOAD
https://github.com/hashgraph/hedera-sdk-js			
Hedera Go SDK	Maintainer: Hedera	License: Apache 2.0	DOWNLOAD
https://github.com/hashgraph/hedera-sdk-go			
Hedera Swift SDK	Maintainer: Hedera	License: Apache 2.0	DOWNLOAD
https://github.com/hashgraph/hedera-sdk-swift			

Development Tools & SDKs

Hedera and the developer community contribute to and maintain developer tools and SDKs. These can be used to make bootstrapping your own project even easier

Additional Language Support

Build a Hedera-powered application using your favorite language with these community-supported SDKs.

[illegible]

<https://github.com/wensheng/hedera-sdk-py>

Ecosystem Wallet Support

If you're building a decentralized application on Hedera that requires wallet connectivity, such as allowing users to connect their HashPack, Blade wallets, or MetaMask, check out the easy-to-implement wallet integration SDKs/tutorials found below. In addition, dApp integration tools for account creation and management, such as Magic Link, make it easy for your users to create an account on Hedera and authenticate themselves in no time.

 Blade Wallet JS API	Maintainer: Community	License: Apache 2.0	https://blade-labs.github.io/blade-web3.js/ DOCUMENTATION
 HashPack HashConnect	Maintainer: Community	License: MIT	https://www.hashpack.app/hashconnect DOCUMENTATION
 Magic Link Wallet SDK	Maintainer: Community	License: MIT	https://magic.link/docs/auth/blockchains/hedera DOCUMENTATION
 MetaMask Integration	Maintainer: Community	License: MIT	https://magic.link/docs/auth/blockchains/hedera
 TUTORIAL			https://tutorials.smart-contracts/create-an-hbar-faucet-app-using-react-and-metamask.md

Decentralized Identity SDKs

Build decentralized identity directly into your Hedera-powered application. Manage DID Documents & a Verifiable Credentials registry, abiding by W3C standards, using the Hedera Consensus Service.

Hedera Java DID SDK	Maintainer: Hedera	License: Apache 2.0	https://github.com/hashgraph/did-sdk-java

https://github.com/hashgraph/did-sdk-java	https://github.com/hashgraph/did-sdk-java
Hedera JavaScript DID SDK	Maintainer: Envision
License: Apache 2.0	https://github.com/hashgraph/did-sdk-js
DOWNLOAD	DOWNLOAD
https://github.com/hashgraph/did-sdk-js	https://github.com/hashgraph/did-sdk-js

Serverless SDKs

Build a Hedera-powered application using your own serverless REST client.

table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th align="center"></th><th align="center"></th><th align="center"></th><th align="center"></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Serverless Hedera REST API</td><td align="center">Maintainer: Community</td><td align="center">License: Apache 2.0</td><td align="center">DOCUMENTATION</td><td align="center"><mark style="color:purple;">DOWNLOAD</mark></td><td></td><td>https://github.com/trustenterprises/hedera-serverless-api</td></tr><tr><td align="center">Serverless Hedera Laravel Package</td><td align="center">Maintainer: Community</td><td align="center">License: Apache 2.0</td><td align="center">DOCUMENTATION</td><td align="center"><mark style="color:purple;">DOWNLOAD</mark></td><td></td><td>https://github.com/trustenterprises/laravel-hashgraph</td></tr></tbody></table>
--

Want to help contribute or have a project you'd like to see, here? Get in touch in discord or add a pull request.

```
# set-up-your-local-network.md:
```

Set Up Your Local Network

While you are developing your application, you can use the Hedera supported networks (previewnet and testnet) to test your application against. In addition to using those networks, you have the option to set-up your own local consensus node and mirror node for testing purposes.\

With your local network set-up you can:

Create and submit transactions and queries to a consensus node
Interact with the mirror node via REST APIs

1. Set Up your local network

Set-up your local network by following the instructions found in the readme of the hedera-local-node project. This will create a Hedera network composed of one consensus node and one mirror node. The consensus node will process incoming transactions and queries. The mirror node stores the history of transactions. Both nodes are created at startup.

2. Configure your network

Once you have your local network up and running, you will need to configure your Hedera client to point to your local network in your project of choice. Your project should have your language specific Hedera SDK as a dependency and imported into your project. You may reference the environment set-up instructions if you don't know how.

Your local network IP address and port will be `127.0.0.1:50211` and your local mirror node IP and port will be `127.0.0.1:5600`. The consensus node account ID is `0.0.3`. This is the node account ID that will receive your transaction and query requests. It is recommended to store these variables in an environment or config file. These values will be hard-coded in the example for demonstration purposes.

Configure your local network by using `Client.forNetwork()`. This allows you to set a custom consensus network by providing the IP address and port. `Client.setMirrorNetwork()` allows you to set a custom mirror node network by providing the IP address and port.

```
{% tabs %}
{% tab title="Java" %}
java
//Create your local client
Client client = Client.forNetwork(Collections.singletonMap("127.0.0.1:50211",
AccountId.fromString("0.0.3"))).setMirrorNetwork(List.of("127.0.0.1:5600"));

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create your local client
const node = {"127.0.0.1:50211": new AccountId(3)};
const client = Client.forNetwork(node).setMirrorNetwork("127.0.0.1:5600");

{% endtab %}

{% tab title="Go" %}
go
//Create your local client
node := make(map[string]hedera.AccountID, 1)
node["127.0.0.1:50211"] = hedera.AccountID{Account: 3}

mirrorNode := []string{"127.0.0.1:5600"}

client := hedera.ClientForNetwork(node)
client.SetMirrorNetwork(mirrorNode)

{% endtab %}
{% endtabs %}
```

3. Set your local node transaction fee paying account

You will need an account ID and key to pay for the fees associated with each transaction and query that is submitted to your local network. You will use the account ID and key provided by the local node on startup to set-up your operator account ID and key. The operator is the default account that pays for transaction and query fees.

```
| Account ID | 0.0.2
|
| ----- |
|-----|
```

```
----- |
| Private Key |
302e020100300506032b65700422042091132178e72057a1d7528025956fe39b0b847f200ab59b2f
dd367017f3087137 |
```

```
{% hint style="danger" %}
```

Note: It is not good practice to post your private keys in any public place. These keys are provided only for development and testing purposes only. They do not exist on any production networks.

```
{% endhint %}
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
client.setOperator(AccountId.fromString("0.0.2"),
PrivateKey.fromString("302e020100300506032b65700422042091132178e72057a1d75280259
56fe39b0b847f200ab59b2fdd367017f3087137"));
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
client.setOperator(AccountId.fromString("0.0.2"),PrivateKey.fromString("302e0201
00300506032b65700422042091132178e72057a1d7528025956fe39b0b847f200ab59b2fdd367017
f3087137"));
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
accountId, err := hedera.AccountIDFromString("0.0.2")
privateKey, err :=
hedera.PrivateKeyFromString("302e020100300506032b65700422042091132178e72057a1d75
28025956fe39b0b847f200ab59b2fdd367017f3087137")
client.SetOperator(accountId, privateKey)
```

```
{% endtab %}
```

```
{% endtabs %}
```

4. Submit your transaction

Submit a transaction that will create a new account in your local network. The console should print out the new account ID. In this example, we are using the same key as the transaction fee paying account as the key for the new account. You can also create a new key if you wish.

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Submit a transaction to your local node
```

```
TransactionResponse newAccount = new AccountCreateTransaction()
    .setKey(PrivateKey.fromString("302e020100300506032b65700422042091132178e
72057a1d7528025956fe39b0b847f200ab59b2fdd367017f3087137"))
    .setInitialBalance(new Hbar(1))
    .execute(client);
```

```
//Get the receipt
```

```
TransactionReceipt receipt = newAccount.getReceipt(client);
```

```
//Get the account ID
```

```
AccountId newAccountId = receipt.accountId;
```

```
System.out.println(newAccountId);
```

```
{% endtab %}
```

```

{% tab title="JavaScript" %}
javascript
//Submit a transaction to your local node
const newAccount = await new AccountCreateTransaction()
    .setKey(PrivateKey.fromString("302e020100300506032b65700422042091132178e
72057a1d7528025956fe39b0b847f200ab59b2fdd367017f3087137"))
    .setInitialBalance(new Hbar(1))
    .execute(client);

//Get receipt
const receipt = await newAccount.getReceipt(client);

//Get the account ID
const newAccountId = receipt.accountId;
console.log(newAccountId);

{% endtab %}

{% tab title="Go" %}
go
//Submit a transaction to your local node
newAccount, err := hedera.NewAccountCreateTransaction().
    SetKey(privateKey).
    SetInitialBalance(hedera.HbarFromTinybar(1000)).
    Execute(client)

if err != nil {
    println(err.Error(), ": error getting balance")
    return
}

//Get receipt
receipt, err := newAccount.GetReceipt(client)

//Get the account ID
newAccountId := receipt.AccountID
fmt.Print(newAccountId)

{% endtab %}
{% endtabs %}

```

5. View your transaction

You can view the executed transaction by querying your local mirror node.

The local mirror node endpoint URL is <http://localhost:5551/>.

You can view the transactions that were submitted to your local node by submitting this request:

```

http
http://localhost:5551/api/v1/transactions

```

The list of supported mirror node REST APIs can be found [here](#). You have now set-up your local environment. Check out the following links for more examples.

```

{% content-ref url="../../tutorials/" %}
tutorials
{% endcontent-ref %}

{% content-ref url="." %}
.

```

{% endcontent-ref %}

Code Check :white\check\mark:

```
{% tabs %}
{% tab title="Java" %}
java
import com.hedera.hashgraph.sdk.;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeoutException;

public class LocalNode {
    public static void main(String[] args) throws TimeoutException,
        PrecheckStatusException, ReceiptStatusException, InterruptedException,
        IOException {

        //Create your local client
        Client client =
        Client.forNetwork(Collections.singletonMap("127.0.0.1:50211",
        AccountId.fromString("0.0.3"))).setMirrorNetwork(List.of("127.0.0.1:5600"));

        //Set the transaction fee paying account
        client.setOperator(AccountId.fromString("0.0.2"),
        PrivateKey.fromString("302e020100300506032b65700422042091132178e72057a1d75280259
        56fe39b0b847f200ab59b2fdd367017f3087137"));

        //Submit a transaction to your local node
        TransactionResponse newAccount = new AccountCreateTransaction()
            .setKey(PrivateKey.fromString("302e020100300506032b6570042204209
        1132178e72057a1d7528025956fe39b0b847f200ab59b2fdd367017f3087137"))
            .setInitialBalance(new Hbar(1))
            .execute(client);

        //Get the receipt
        TransactionReceipt receipt = newAccount.getReceipt(client);

        //Get the account ID
        AccountId newAccountId = receipt.accountId;
        System.out.println(newAccountId);
    }
}

{% endtab %}

{% tab title="JavaScript" %}
javascript
const {
    Client,
    PrivateKey,
    Hbar,
    AccountId,
    AccountCreateTransaction,
} = require("@hashgraph/sdk");

async function main() {

    //Create your local client
    const node = {"127.0.0.1:50211": new AccountId(3)}
    const client =
    Client.forNetwork(node).setMirrorNetwork("127.0.0.1:5600");

    //Set the transaction fee paying account
```

```

client.SetOperator(AccountId.fromString("0.0.2"), PrivateKey.fromString("302e0201
00300506032b65700422042091132178e72057a1d7528025956fe39b0b847f200ab59b2fdd367017
f3087137"));

    //Submit a transaction to your local node
    const newAccount = await new AccountCreateTransaction()
        .setKey(PrivateKey.fromString("302e020100300506032b6570042204209
1132178e72057a1d7528025956fe39b0b847f200ab59b2fdd367017f3087137"))
        .setInitialBalance(new Hbar(1))
        .execute(client);

    //Get receipt
    const receipt = await newAccount.getReceipt(client);

    //Get the account ID
    const newAccountId = receipt.accountId;
    console.log(newAccountId);
}
void main();

{% endtab %}

{% tab title="Go" %}
go
package main

import (
    "fmt"

    "github.com/hashgraph/hedera-sdk-go/v2"
)

func main() {
    //Create your local node client
    node := make(map[string]hedera.AccountID, 1)
    node["127.0.0.1:50211"] = hedera.AccountID{Account: 3}

    mirrorNode := []string{"127.0.0.1:5600"}

    client := hedera.ClientForNetwork(node)
    client.SetMirrorNetwork(mirrorNode)

    //Set the transaction fee paying account
    accountId, err := hedera.AccountIDFromString("0.0.2")
    privateKey, err :=
hedera.PrivateKeyFromString("302e020100300506032b65700422042091132178e72057a1d75
28025956fe39b0b847f200ab59b2fdd367017f3087137")
    client.SetOperator(accountId, privateKey)

    //Submit a transaction to your local node
    newAccount, err := hedera.NewAccountCreateTransaction().
        SetKey(privateKey).
        SetInitialBalance(hedera.HbarFromTinybar(1000)).
        Execute(client)

    if err != nil {
        println(err.Error(), ": error getting balance")
        return
    }

    //Get receipt
    receipt, err := newAccount.GetReceipt(client)

```

```

        //Get the account ID
        newAccountId := receipt.AccountID
        fmt.Print(newAccountId)
    }

{% endtab %}
{% endtabs %}

```

specialized-types.md:

Specialized Types

AccountId

An AccountId is composed of a \<shardNum>.\<realmNum>.\<accountNum> (eg. 0.0.10).

Shard number (shardNum) represents the shard number (shardId). It will default to 0 today, as Hedera only performs in one shard.

Realm number (realmNum) represents the realm number (realmId). It will default to 0 today, as realms are not yet supported.

Account represents either an account number or an account alias

Account number (accountNum) represents the account number (accountId)

Account alias (alias) represented by the public key bytes

The public key bytes are the result of serializing a protobuf Key message for any primitive key type

Currently, only primitive key bytes are supported as an alias

Threshold keys, key list, contract ID, and delegatable\contract\id are not supported

The alias can only be used in place of an account ID in transfer transactions in its current version

Together these values make up your AccountId. When an AccountId is specified, be sure all three values are included.

Constructor

Constructor	
Type	
<code>new AccountId(&#x3C;shardNum>,&#x3C;realmNum>,&#x3C;accountNum>)</code>	
long, long, long	Constructs an <code>AccountId</code> with 0 for <code>shardNum</code> and <code>realmNum</code> (e.g., <code>0.0.&#x3C;accountNum></code>)

Methods

Methods	
Type	
<code>AccountId.fromString(&#x3C;account>)</code>	
String	Constructs an <code>AccountId</code> from a string formatted as <code>&#x3C;shardNum>.&#x3C;realmNum>.&#x3C;accountNum></code>
<code>AccountId.fromEvmAddress(&#x3C;address>)</code>	
String	Constructs an <code>AccountId</code> from a solidity address in string format
<code>AccountId.fromBytes(bytes)</code>	
byte[]	Constructs an <code>AccountId</code> from bytes
<code>AccountId.toSolidityAddress()</code>	
String	Constructs a solidity address from

<code>AccountID</code>	
<code>AccountID.toString()</code>	String Constructs an <code>AccountID</code> from string
<code>AccountID.aliasKey</code>	PublicKey The alias key of the <code>AccountID</code>
<code>AccountID.aliasEvmAddress</code>	EVM address The EVM address of the <code>AccountID</code>
<code>AccountID.toBytes()</code>	byte[] Constructs an <code>AccountID</code> from bytes

Example

```
{% tabs %}
{% tab title="Java" %}
java
AccountId accountId = new AccountId(0 ,0 ,10);
System.out.println(accountId);

// Constructs an accountId from String
AccountId accountId = AccountId.fromString("0.0.10");
System.out.println(accountId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
const accountId = new AccountId(100);
console.log(`${accountId}`);

// Construct accountId from String
const accountId = AccountId.fromString(100);
console.log(`${accountId}`);

{% endtab %}

{% tab title="Go" %}
go
hedera.AccountIDFromString("0.0.3")

{% endtab %}
{% endtabs %}
```

FileId

A FileId is composed of a \<shardNum>.\<realmNum>.\<fileNum> (eg. 0.0.15).

shardNum represents the shard number (shardId). It will default to 0 today, as Hedera only performs in one shard.

realmNum represents the realm number (realmId). It will default to 0 today, as realms are not yet supported.

fileNum represents the file number

Together these values make up your accountId. When an FileId is requested, be sure all three values are included.

Constructor

Constructor	
Type	Description
<code>new FileId(&#x3C;shardNum>,&#x3C;realmNum>,&#x3C;fileNum>)</code>	long, long, long Constructs a <code>FileId</code> with 0

for `<shardNum>` and `<realmNum>` (e.g.,
`<0.0.<fileNum>`)

Methods

Meth	Type
Description	
<code>FileId.fromString()</code>	String
Constructs an <code>FileId</code> from a string formatted as <code><shardNum>.<realmNum>.<fileNum></code>	
<code>FileId.fromSolidityAddress()</code>	String
Constructs an <code>FileId</code> from a solidity address in string format	
<code>FileId.ADDRESSBOOK</code>	FileId
The public node address book for the current network	
<code>FileId.EXCHANGERATES</code>	FileId
The current exchange rate of HBAR to USD	
<code>FileId.FEESCHEDULE</code>	FileId
The current fee schedule for the network	

Example

```
{% tabs %}
{% tab title="Java" %}
java
FileId fileId = new FileId(0,0,15);
System.out.println(fileId);

//Constructs a FileId from string
FileId fileId = FileId.fromString("0.0.15");
System.out.println(fileId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
const newFileId = new FileId(100);
console.log(`${newFileId}`);

//Construct a fileId from a String
const newFileIdFromString = FileId.fromString(100);
console.log(`${newFileIdFromString}`);

{% endtab %}

{% tab title="Go" %}
go
hedera.FileIDFromString("0.0.3")

{% endtab %}
{% endtabs %}
```

ContractId

A `ContractId` is composed of a `<shardNum>.<realmNum>.<contractNum>` (eg. `0.0.20`).

`shardNum` represents the shard number (`shardId`). It will default to 0 today, as Hedera only performs in one shard.

`realmNum` represents the realm number (`realmId`). It will default to 0 today, as realms are not yet supported.

`contractNum` represents the contract number

Together these values make up your `ContractId`. When an `ContractId` is requested,

be sure all three values are included. ContractId's are automatically assigned when you create a new smart contract.

Constructor

Constructor	Type	Description
new ContractId(<shardNum>,<realmNum>,<contractNum>)	long, long, long	Constructs a ContractId with 0 for shardNum and realmNum (e.g., 0.0.<contractNum>)

Methods

<code>ContractId.fromString(&#x3C;account>)</code>	String	Constructs a <code>ContractId</code> from a string formatted as <code>&#x3C;shardNum>.&#x3C;realmNum>.&#x3C;contractNum></code>
<code>ContractId.fromSolidityAddress(&#x3C;address>)</code> [deprecated use <code>ContractId.fromEvmAddress()</code>]	String	Constructs a <code>ContractId</code> from a solidity address in string format [deprecated use <code>ContractId.fromEvmAddress()</code>]
<code>ContractId.toSolidityAddress(&#x3C;contractId>)</code>	String	Construct a Solidity address from a Hedera contract ID
<code>ContractId.fromEvmAddress(&#x3C;shard>, &#x3C;realm>, &#x3C;evmAddress>)</code>	long, long, String	Constructs a <code>ContractId</code> from evm address

Example

```
{% tabs %}
{% tab title="Java" %}
java
ContractId contractId = new ContractId(0,0,20);
System.out.println(contractId);

// Constructs a ContractId from string
ContractId contractId = ContractId.fromString("0.0.20");
System.out.println(contractId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
const newContractId = new ContractId(100);
console.log(`${newContractId}`);

// Construct a contractId from a String
const newContractId = ContractId.fromString(100);
console.log(`${newContractId}`);

{% endtab %}

{% tab title="Go" %}

hedera.ContractIDFromString("0.0.3")

{% endtab %}
{% endtabs %}
```

TopicId

A topicId is composed of a \<shardNum>.\<realmNum>.\<topicNum> (eg. 0.0.100).

shardNum represents the shard number (shardId). It will default to 0 today, as Hedera only performs in one shard.

realmNum represents the realm number (realmId). It will default to 0 today, as realms are not yet supported.

topicNum represents the topic number (topicId)

Constructor

Constructor	Type
Description	
<code>new ConsensusTopicId(&#x3C;shardNum>,&#x3C;realmNum>,&#x3C;topicNum>)</code>	<code>long, long, long</code>
Constructs a <code>TopicId</code> with <code>0</code> for <code>shardNum</code> and <code>realmNum</code> (e.g., <code>0.0.&#x3C;topicNum></code>)	

Methods	Type
Description	
<code>fromString(&#x3C;topic>)</code>	<code>String</code>
Constructs a topic ID from a <code>String</code>	
<code>ConsensusTopicId.toString()</code>	
Constructs a topic ID to String format	

Example

```
{% tabs %}
{% tab title="Java" %}
java
ConsensusTopicId topicId = new ConsensusTopicId(0,0,100);
System.out.println(topicId)
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
const topicId = new ConsensusTopicId(0,0,100);
console.log(topicId)
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
hedera.TopicIDFromString("0.0.3")
```

```
{% endtab %}
{% endtabs %}
```

adjust-an-allowance.md:

Delete an allowance

A transaction that deletes one or more non-fungible approved allowances from an owner's account. This operation will remove the allowances granted to one or more specific non-fungible token serial numbers. Each owner account listed as wiping an allowance must sign the transaction. HBAR and fungible token allowances can be removed by setting the amount to zero in

CryptoApproveAllowance.

The total number of NFT serial number deletions within the transaction body cannot exceed 20.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Transaction Signing Requirements

The transaction must be signed by the owner's account
The transaction must be signed by the transaction fee-paying account if different than the owner's account
If the owner's account and transaction fee-paying account are the same, only one signature is required

Reference: HIP-336

Methods

Method	Type
Description	
-----	-----
deleteAllTokenNftAllowances(<nftId>, <ownerAccountId>)	<p>NFT ID, AccountId</p> Removes the NFT allowance from the spender account.

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
AccountAllowanceDeleteTransaction transaction = new
AccountAllowanceDeleteTransaction()
    .deleteAllTokenNftAllowances(nftId , ownerAccountId);

//Sign the transaction with the owner account key
TransactionResponse txResponse =
transaction.freezeWith(client).sign(ownerAccountKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.12.0+

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new AccountAllowanceDeleteTransaction()
    .deleteAllTokenNftAllowances(nftId , ownerAccountId);

//Sign the transaction with the owner account key
const signTx = await transaction.sign(ownerAccountKey);
```

```

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

//v2.13.0+

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewAccountAllowanceDeleteTransaction().
    DeleteAllTokenNftAllowances(nftId , ownerAccountId)

if err != nil {
    panic(err)
}

//Sign the transaction with the owner account private key and submit to the
network
txResponse, err := transaction.Sign(ownerAccountKey).Execute(client)

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

println("The transaction consensus status is ", transactionStatus)

//v2.13.1+

{% endtab %}
{% endtabs %}

```

approve-an-allowance.md:

Approve an allowance

A transaction that allows a token owner to delegate a token spender to spend the specified token amount on behalf of the token owner. A Hedera account owner can provide an allowance for HBAR, non-fungible, and fungible tokens.

The owner is the Hedera account that owns the tokens and grants the token allowance to the spender. The spender is the account that spends tokens, authorized by the owner, from the owner's account. The spender pays for the transaction fees when transferring tokens from the owner's account to another recipient. This means that the transaction fee payer for the TransferTransaction is required to set the spender account ID as the transaction fee payer. If the spender account ID is not set as the transaction fee payer, the system will error with SPENDERDOESNOTHAVEALLOWANCE.

The maximum number of token approvals for the AccountAllowanceApproveTransaction cannot exceed 20. Note that each NFT serial number counts as a single approval. An AccountAllowanceApproveTransaction granting 20 NFT serial numbers to a spender will use all of the approvals permitted for the transaction.

A single NFT serial number can only be granted to one spender at a time. If an approval assigns a previously approved NFT serial number to a new user, the old user will have their approval removed.

Each owner account is limited to granting 100 allowances. This limit spans HBAR, fungible token allowances, and non-fungible token approvedforall grants. No limit exists on the number of NFT serial number approvals an owner may grant.

The number of allowances set on an account will increase the auto-renewal fee for the account. Conversely, removing allowances will decrease the auto-renewal fee for the account.

To decrease the allowance for a given spender, you must set the amount to the value you would like to authorize the account for. If the spender account was authorized to spend 25 HBAR and the owner wants to modify their allowance to 5 HBAR, the owner would submit the AccountAllowanceApproveTransaction for 5 HBAR.

Only when a spender is set on an explicit NFT ID of a token, do we return the spender ID in TokenNftInfoQuery for the respective NFT. If approveTokenNftAllowanceAllSerials is used to approve all NFTs for a given token class, and no NFT ID is specified; we will not return a spender ID for all the serial numbers of that token.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Transaction Signing Requirements

Must be signed by the owner's account
Must be signed by the transaction fee payer if different then the owner account
If the owner and transaction fee payer key are the same only one signature is required

Reference: HIP-336

Methods

Method	Type	Description
approveHbarAllowance(<ownerAccountId>,<spenderAccountId>, <amount>)	AccountId, AccountId, Hbar	The owner account ID that is authorizing the allowance, the spender account ID to authorize, the amount of hbar the owner account is authorizing the spender account to use.
approveTokenAllowance(<tokenId>,<ownerAccountId>,<spenderAccountId>, <amount>)		

```
| <p><a href="../token-service/token-id.md">TokenId</a>,<br><a href="../specialized-types.md#accountid">AccountId</a>,</p><p><a href="../specialized-types.md#accountid">AccountId</a>,</p> | The token ID of the token being granted an allowance by the spender account, the account ID of the owner account, the account ID of the spender account.
|
| approveTokenNftAllowance(<nftId>,<ownerAccountId>,<spenderAccountId>)
| <p><a href="../token-service/nft-id.md">nftId</a>,<a href="../specialized-types.md#accountid">AccountId</a>,<br><a href="../specialized-types.md#accountid">AccountId</a></p> | The NFT ID of the NFT being granted an allowance by the owner account, the account ID of the owner account, the account ID of the spender account.
|
| approveTokenNftAllowanceAllSerials(<tokenId>,<ownerAccountId>,<spenderAccountId>) | <p><a href="../token-service/token-id.md">TokenId</a>,<br><a href="../specialized-types.md#accountid">AccountId</a>,<br><a href="../specialized-types.md#accountid">AccountId</a>,</p> | Grant a spender account access to all NFTs in a given token class/collection. The token ID of the NFT collection, the account ID of the owner account, the account ID of the spender account. |
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
AccountAllowanceApproveTransaction transaction = new
AccountAllowanceApproveTransaction()
    .approveHbarAllowance(ownerAccount, spenderAccountId, Hbar.from(1));

//Sign the transaction with the owner account key and the transaction fee payer
key (client)
TransactionResponse txResponse =
transaction.freezeWith(client).sign(ownerAccountKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.12.0+

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new AccountAllowanceApproveTransaction()
    .approveHbarAllowance(ownerAccount, spenderAccountId, Hbar.from(1));

//Sign the transaction with the owner account key
const signTx = await transaction.sign(ownerAccountKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
```

```

const transactionStatus = receipt.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

//v2.13.0

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewAccountAllowanceApproveTransaction().
    ApproveHbarAllowance(ownerAccount, spenderAccountId, Hbar.fromTinybars(1))
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign the transaction with the owner account private key
txResponse, err := transaction.Sign(ownerAccountKey).Execute(client)

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

println("The transaction consensus status is ", transactionStatus)
//v2.13.1+

{% endtab %}
{% endtabs %}

```

create-an-account.md:

Create an account

Create an account using the account create API

A transaction that creates a Hedera account. A Hedera account is required to interact with any of the Hedera network services as you need an account to pay for all associated transaction/query fees. You can visit the Hedera Developer Portal to create a previewnet or testnet account. You can also use third-party wallets to generate free mainnet accounts. To process an account create transaction, you will need an existing account to pay for the transaction fee. To obtain the new account ID, request the receipt of the transaction.

```

{% hint style="info" %}
When creating a new account using the<mark
style="color:purple;">AccountCreateTransaction()</mark>API you will need an
existing account to pay for the associated transaction fee.
{% endhint %}

```

Account Properties

```

{% content-ref url="../../../core-concepts/accounts/account-properties.md" %}
account-properties.md
{% endcontent-ref %}

```


Transaction Fees

The sender pays for the token association fee and the rent for the first auto-renewal period.

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate your transaction fee cost.

Transaction Signing Requirements

The account paying for the transaction fee is required to sign the transaction.

Maximum Auto-Associations and Fees

Accounts have a property, `maxAutoAssociations`, and the property's value determines the maximum number of automatic token associations allowed.

Property Value	Description
<code>0</code>	Automatic token associations or <a data-footnote-ref="" href="#user-content-fn-1">token airdrops are not allowed, and the account must be manually associated with a token. This also applies if the value is less than or equal to <code>usedAutoAssociations</code> .
<code>-1</code>	The number of automatic token associations an account can have is unlimited. <code>-1</code> is the default value for new automatically-created accounts.
<code>> 0</code>	If the value is a positive number (number greater than 0), the number of automatic token associations an account can have is limited to that number.

{% hint style="info" %}

The sender pays the `maxAutoAssociations` fee and the rent for the first auto-renewal period for the association. This is in addition to the typical transfer fees. This ensures the receiver can receive tokens without association and makes it a smoother transfer process.

{% endhint %}

Reference: HIP-904

Methods

Method	Type	Requirement
<code>setKey(&#x3C;key>)</code>	Key	Required
<code>setAlias(&#x3C;alias>)</code>	EvmAddress	Optional
<code>setInitialBalance(&#x3C;initialBalance>)</code>	HBar	Optional
<code>setReceiverSignatureRequired(&#x3C;booleanValue>)</code>	boolean	Optional
<code>setMaxAutomaticTokenAssociations(&#x3C;amount>)</code>	int	Optional
<code>setStakedAccountId(&#x3C;stakedAccountId>)</code>	AccountId	Optional
<code>setStakedNodeId(&#x3C;stakedNodeId>)</code>	long	Optional
<code>setDeclineStakingReward(&#x3C;declineStakingReward>)</code>	boolean	Optional
<code>setAccountMemo(&#x3C;memo>)</code>	String	Optional
<code>setAutoRenewPeriod(&#x3C;autoRenewPeriod>)</code>	Duration	Disabled

```

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
AccountCreateTransaction transaction = new AccountCreateTransaction()
    .setKey(privateKey.getPublicKey())
    .setInitialBalance(new Hbar(1000));

//Submit the transaction to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the account ID
AccountId newAccountId = receipt.accountId;

System.out.println("The new account ID is " +newAccountId);

//Version 2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new AccountCreateTransaction()
    .setKey(privateKey.publicKey)
    .setInitialBalance(new Hbar(1000));

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the account ID
const newAccountId = receipt.accountId;

console.log("The new account ID is " +newAccountId);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewAccountCreateTransaction().
    SetKey(privateKey.PublicKey()).
    SetInitialBalance(hedera.NewHbar(1000))

//Sign the transaction with the client operator private key and submit to a
Hedera network
txResponse, err := AccountCreateTransaction.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {

```

```

    panic(err)
}

//Get the account ID
newAccountId := receipt.AccountID

fmt.Printf("The new account ID is %v\n", newAccountId)

//Version 2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method                                | Type      | Description
| -----|-----|-----
| getKey()                             | Key       | Returns the public key on the
account
| getInitialBalance()                  | Hbar      | Returns the initial balance of
the account
| getAutoRenewPeriod()                 | Duration  | Returns the auto renew period on
the account
| getDeclineStakingReward()            | boolean   | Returns whether or not the
account declined rewards
| getStakedNodeId()                   | long      | Returns the node ID
| getStakedAccountId()                | AccountId | Returns the node account ID
| getReceiverSignatureRequired()      | boolean   | Returns whether the receiver
signature is required or not

{% tabs %}
{% tab title="Java" %}
java
//Create an account with 1,000 hbar
AccountCreateTransaction transaction = new AccountCreateTransaction()
    // The only required property here is key
    .setKey(newKey.getPublicKey())
    .setInitialBalance(new Hbar(1000));

//Return the key on the account
Key accountKey = transaction.getKey();

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create an account with 1,000 hbar
const transaction = new AccountCreateTransaction()
    // The only required property here is key
    .setKey(newKey.getPublicKey())
    .setInitialBalance(new Hbar(1000));

//Return the key on the account
const accountKey = transaction.getKey();

{% endtab %}

{% tab title="Go" %}
go
//Create an account with 1,000 hbar

```

```
AccountCreateTransaction := hedera.NewAccountCreateTransaction().
    SetKey(newKey.PublicKey()).
    SetInitialBalance(hedera.NewHbar(1000))
```

```
//Return the key on the account
accountKey, err := AccountCreateTransaction.GetKey()
```

```
{% endtab %}
{% endtabs %}
```

```
[^1]:
```

```
# delete-an-account.md:
```

```
Delete an account
```

A transaction that deletes an existing account from the Hedera network. Before deleting an account, the existing HBAR must be transferred to another account. Submitting a transaction to delete an account without assigning a beneficiary via `setTransferAccountId()` will result in a `ACCOUNTIDDOESNOTEXIST` error. Transfers cannot be made into a deleted account. A record of the deleted account will remain in the ledger until it expires. The expiration of a deleted account can be extended. The account that is being deleted is required to sign the transaction.

```
{% hint style="info" %}
```

Note: The `setTransferAccountId()` method is required, regardless of whether the account has a zero balance.

```
{% endhint %}
```

```
Transaction Fees
```

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate your transaction fee cost.

```
Transaction Signing Requirements
```

The account that is being deleted is required to sign the transaction.

```
Methods
```

Method	Type	Description	Requirement
<code>setAccountId(&#x3C;accountId>)</code>		AccountId	The ID of the account to delete.
<code>setTransferAccountId(&#x3C;transferAccountId>)</code>		AccountId	The ID of the account to transfer the remaining funds to.

```
{% tabs %}
{% tab title="Java" %}
```

```
java
//Create the transaction to delete an account
AccountDeleteTransaction transaction = new AccountDeleteTransaction()
    .setAccountId(accountId)
    .setTransferAccountId(OPERATORID);
```

```
//Freeze the transaction for signing, sign with the private key of the account
that will be deleted, sign with the operator key and submit to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(newKey).execute(client);
```

```
//Request the receipt of the transaction
```

```

TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction to delete an account
const transaction = await new AccountDeleteTransaction()
    .setAccountId(accountId)
    .setTransferAccountId(OPERATORID)
    .freezeWith(client);

//Sign the transaction with the account key
const signTx = await transaction.sign(accountKey);

//Sign with the client operator private key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction to delete an account, freeze the transaction for
signing
transaction, err := hedera.NewAccountDeleteTransaction().
    SetAccountID(newAccountID).
    SetTransferAccountID(operatorAccountID).
    FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the private key of the account that will be deleted, sign with the
operator key and submit to a Hedera network
txResponse, err := transaction.Sign(accountKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", transactionStatus)

```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

Get transaction values

```
<table><thead><tr><th width="370.3333333333333">Method</th><th  
width="124.7980179329873">Type</th><th>Description</th></tr></thead><tbody><tr><  
td><code>getAccountId(&#x3C;accountId>)</code></td><td>AccountId</td><td>The  
account to  
delete</td></tr><tr><td><code>getTransferAccountId(&#x3C;transferAccountId>)</  
code></td><td>AccountId</td><td>The account to transfer the remaining funds  
to</td></tr></tbody></table>
```

```
{% tabs %}  
{% tab title="Java" %}
```

```
java  
//Create the transaction to delete an account  
AccountDeleteTransaction transaction = new AccountDeleteTransaction()  
    .setAccountId(newAccountId)  
    .setTransferAccountId(OPERATORID);  
  
//Get the account ID from the transaction  
AccountId transactionAccountId = transaction.getAccountId()  
  
System.out.println("The account to be deleted in this transaction is "  
+transactionAccountId)
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java  
//Create the transaction to delete an account  
const transaction = new AccountDeleteTransaction()  
    .setAccountId(newAccountId)  
    .setTransferAccountId(OPERATORID);  
  
//Get the account ID from the transaction  
const transactionAccountId = transaction.getAccountId()  
  
console.log("The account to be deleted in this transaction is "  
+transactionAccountId)
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java  
//Create the transaction to delete an account  
transaction, err := hedera.NewAccountDeleteTransaction().  
    SetAccountId(newAccountId).  
    SetTransferAccountId(operatorAccountId)
```

```
//Get the account ID from the transaction  
transactionAccountId := transaction.GetAccountId()
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

errors.md:

Network Response Messages

Network response messages and their descriptions.

Errors	Description
-----	-----
ACCOUNTIDDOESNOTEXIST passed has not yet been created.	The account id
ACCOUNTUPDATEFAILED the account failed	The update of
ACCOUNTDELETED has been marked as deleted	The account
INVALIDACCOUNTAMOUNTS transfer credit and debit do not sum equal to 0	The crypto
INVALIDINITIALBALANCE the negative initial balance	Attempt to set
INVALIDRECEIVERRECORDTHRESHOLD negative receive record threshold	Attempt to set
INVALIDSENDRECORDTHRESHOLD negative send record threshold	Attempt to set
SETTINGNEGATIVEACCOUNTBALANCE set negative balance value for the crypto account	Attempting to
TRANSFERLISTSIZELIMITEXCEEDED number of accounts (both from and to) allowed for crypto transfer list	Exceeded the
TRANSFERACCOUNTSAMEASDELETEACCOUNT should not be same as Account to be deleted	Transfer Account
NOREMAININGAUTOMATICASSOCIATIONS reached the limit on the automatic associations count.	The account has
EXISTINGAUTOMATICASSOCIATIONSEXCEEDGIVENLIMIT automatic associations are more than the new maximum automatic associations.	Already existing
REQUESTEDNUMAUTOMATICASSOCIATIONSEXCEEDSASSOCIATIONLIMIT number of automatic associations for an account more than the maximum allowed token associations tokens.maxPerAccount	Cannot set the

get-account-balance.md:

Get account balance

A query that returns the account balance for the specified account. Requesting an account balance is currently free of charge. Queries do not change the state of the account or require network consensus. The information is returned from a single node processing the query.

In Services release 0.50, returning token balance from the consensus node was deprecated with HIP-367. This query returns token information by requesting the

information from the Hedera Mirror Node APIs via `/api/v1/accounts/{id}/tokens`.
Token symbol is not returned in the response.

Query Fees

Please see the transaction and query fees table for the base transaction fee.
Please use the Hedera fee estimator to estimate your query fee cost.

Query Signing Requirements

The client operator private key is required to sign the query request.

Methods

Method	Type	Description
<code>setAccountId(&#x3C;accountId>)</code>	AccountID	The account ID to return the current balance for.
<code>setContractId(&#x3C;contractId>)</code>	ContractID	The contract ID to return the current balance for.

```
{% tabs %}
{% tab title="Java" %}
java
//Create the account balance query
AccountBalanceQuery query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Sign with client operator private key and submit the query to a Hedera network
AccountBalance accountBalance = query.execute(client);

//Print the balance of hbars
System.out.println("The hbar account balance for this account is "
+accountBalance.hbars);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the account balance query
const query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Submit the query to a Hedera network
const accountBalance = await query.execute(client);

//Print the balance of hbars
console.log("The hbar account balance for this account is "
+accountBalance.hbars);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the account balance query
query := hedera.NewAccountBalanceQuery().
    SetAccountID(newAccountId)

//Sign with client operator private key and submit the query to a Hedera network
```



```

accountBalance, err := query.Execute(client)
if err != nil {
    panic(err)
}

```

```

//Print the balance of hbars
fmt.Println("The hbar account balance for this account is ",
accountBalance.Hbars.String())
//v2.0.0

```

```

{% endtab %}
{% endtabs %}

```

get-account-info.md:

Get account info

A query that returns the current state of the account. This query does not include the list of records associated with the account. Anyone on the network can request account info for a given account. Queries do not change the state of the account or require network consensus. The information is returned from a single node processing the query.

In Services release 0.50, returning token balance information from the consensus node was deprecated with HIP-367. This query now returns token information by requesting the information from the Hedera Mirror Node APIs via `/api/v1/accounts/{id}/tokens`. Token symbol is not returned in the response.\

Account Properties

```

{% content-ref url="../../../core-concepts/accounts/account-properties.md" %}
account-properties.md
{% endcontent-ref %}

```

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Query Signing Requirements

The client operator private key is required to sign the query request.

Methods

Method	Type	Requirement
<code>setAccountId(<#x3C;accountId>)</code>	AccountId	Required
<code><#x3C;AccountInfo>.accountId</code>	AccountId	Optional
<code><#x3C;AccountInfo>.contractAccountId</code>	String	Optional
<code><#x3C;AccountInfo>.isDeleted</code>	boolean	Optional
<code><#x3C;AccountInfo>.key</code>	Key	Optional
<code><#x3C;AccountInfo>.balance</code>	HBAR	Optional
<code><#x3C;AccountInfo>.isReceiverSignatureRequired</code>	boolean	Optional
<code><#x3C;AccountInfo>.ownedNfts</code>	long	Optional
<code><#x3C;AccountInfo>.maxAutomaticTokenAssociations</code>	int	Optional
<code><#x3C;AccountInfo>.accountMemo</code>	String	Optional
<code><#x3C;AccountInfo>.expirationTime</code>		

code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>#x3C;AccountInfo>.autoRenewPeriod</code></td><td>Duration</td><td>Optional</td></tr><tr><td><code>#x3C;AccountInfo>.ledgerId</code></td><td>LedgerId</td><td>Optional</td></tr><tr><td><code>#x3C;AccountInfo>.ethereumNonce</code></td><td>long</td><td>Optional</td></tr><tr><td><code>#x3C;AccountInfo>.stakingInfo</code></td><td>Optional</td><td></td></tr><tr><td><code>#x3C;AccountInfo>.tokenRelationships</code></td><td>Map#x3C;TokenId, TokenRelationships></td><td>Optional</td></tr><tr><td></td><td></td><td></td></tr></tbody></table>
--

```
{% tabs %}
{% tab title="Java" %}
java
//Create the account info query
AccountInfoQuery query = new AccountInfoQuery()
    .setAccountId(newAccountId);

//Submit the query to a Hedera network
AccountInfo accountInfo = query.execute(client);

//Print the account key to the console
System.out.println(accountInfo);

//v2.0.0

{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create the account info query
const query = new AccountInfoQuery()
    .setAccountId(newAccountId);

//Sign with client operator private key and submit the query to a Hedera network
const accountInfo = await query.execute(client);

//Print the account info to the console
console.log(accountInfo);

//v2.0.0

{% endtab %}
```

```
{% tab title="Go" %}
go
//Create the account info query
query := hedera.NewAccountInfoQuery().
    SetAccountID(newAccountId)

//Sign with client operator private key and submit the query to a Hedera network
accountInfo, err := query.Execute(client)
if err != nil {
    panic(err)
}
```

```
//Print the account info to the console
fmt.Println(accountInfo)
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```


Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Spender Account Allowances

An account can have another account spend tokens on its behalf. If the delegated spender account is transacting tokens from the owner account that authorized the allowance, the owner account needs to be specified in the transfer transaction by calling one of the following:

```
addApprovedHbarTransfer()  
addApprovedTokenTransfer()  
addApprovedNftTransfer()  
addApprovedTokenTransferWithDecimals()
```

The debiting account is the owner's account when using this feature.

{% hint style="info" %}

Note: The allowance spender must pay the fee for the transaction.

{% endhint %}

Transaction Signing Requirements

The accounts the tokens are being debited from are required to sign the transaction

If an authorized spender account is spending on behalf of the account that owns the tokens then the spending account is required to sign

The transaction fee-paying account is required to sign the transaction

Methods

Method	Type	Description
<code>addHbarTransfer(<accountId>, <evmAddress>, <value>)</code>	AccountId, string, HBAR	The account involved in the transfer and the number of HBAR. The sender and recipient values must net zero.
<code>addTokenTransfer(<tokenId>, <accountId>, <evmAddress>, <value>)</code>	TokenId, AccountId, string, long	The ID of the token, the account ID involved in the transfer, and the number of tokens to transfer. The sender and recipient values must net zero.
<code>addNftTransfer(<nftId>, <sender>, <receiver>)</code>	NftId, AccountId, AccountId	The NFT ID (token + serial number), the sending account, and receiving account.
<code>addTokenTransferWithDecimals(<tokenId>, <accountId>, <value>, <int>)</code>	TokenId, AccountId, long, decimals	The ID of the token, the account ID involved in the transfer, the number of tokens to transfer, the decimals of the token. The sender and recipient values must net zero.
<code>addApprovedHbarTransfer(<ownerAccountId>, <amount>)</code>	AccountId, Hbar	The owner account ID the spender is authorized to transfer from and the amount. Applicable to allowance transfers only.
<code>addApprovedTokenTransfer(<tokenId>, <accountId>, <value>)</code>	TokenId, AccountId, long	The owner account ID and token the spender is authorized to transfer from. The debiting account is the owner account. Applicable to allowance transfers

only.	
<code>addApprovedTokenTransferWithDecimals(&#x3C;tokenId>, &#x3C;accountId>, &#x3C;value>, &#x3C;decimals>)</code>	TokenId , AccountId , long, int
The owner account ID and token ID (with decimals) the spender is authorized to transfer from. The debit account is the account ID of the sender.	Applicable to allowance transfers only.
<code>addApprovedNftTransfer(&#x3C;nftId>, &#x3C;sender>, &#x3C;receiver>)</code>	NftId , AccountId , AccountId
The NFT ID the spender is authorized to transfer. The sender is the owner account and receiver is the receiving account.	Applicable to allowance transfers only.

```
{% tabs %}
{% tab title="Java" %}
java
// Create a transaction to transfer 1 HBAR
TransferTransaction transaction = new TransferTransaction()
    .addHbarTransfer(OPERATORID, new Hbar(-1))
    .addHbarTransfer(newAccountId, evmAddress, new Hbar(1));

//Submit the transaction to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//Version 2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create a transaction to transfer 1 HBAR
const transaction = new TransferTransaction()
    .addHbarTransfer(OPERATORID, new Hbar(-1))
    .addHbarTransfer(newAccountId, evmAddress, new Hbar(1));

//Submit the transaction to a Hedera network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
// Create a transaction to transfer 1 HBAR
transaction := hedera.NewTransferTransaction().
```

```

        AddHbarTransfer(client.GetOperatorAccountID(), hedera.NewHbar(-1)).
        AddHbarTransfer(hedera.AccountID{Account: 3}, hedera.NewHbar(1))

//Submit the transaction to a Hedera network
txResponse, err := transaction.Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status is %v\n",
transactionReceipt.Status)

//Version 2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method | Type | Description
| ----- | ----- | -----
| getHbarTransfers() | Map<AccountID, Hbar> | Returns a list of
the hbar transfers in this transaction |
| getTokenTransfers() | Map<TokenID, Map<AccountID, long>> | Returns the list
of token transfers in the transaction |

{% tabs %}
{% tab title="Java" %}
java
// Create a transaction
CryptoTransferTransaction transaction = new CryptoTransferTransaction()
    .addSender(OPERATORID, new Hbar(1))
    .addRecipient(newAccountId, new Hbar(1));

//Get transfers
List<Transfer> transfers = transaction.getTransfers();

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create a transaction
const transaction = new CryptoTransferTransaction()
    .addSender(OPERATORID, new Hbar(1))
    .addRecipient(newAccountId, new Hbar(1));

//Get transfers
const transfers = transaction.getTransfers();

```

```
//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
// Create a transaction
transaction := hedera.NewTransferTransaction().
    AddHbarTransfer(client.GetOperatorAccountID(), hedera.NewHbar(-1)).
    AddHbarTransfer(hedera.AccountID{Account: 3}, hedera.NewHbar(1))
//Get transfers
transfers := transaction.GetTransfers()

//v2.0.0

{% endtab %}
{% endtabs %}
```

update-an-account.md:

Update an account

A transaction that updates the properties of an existing account. The network will store the latest updates on the account. If you would like to retrieve the state of an account in the past, you can query a mirror node.

Account Properties

```
{% content-ref url="../../../core-concepts/accounts/account-properties.md" %}
account-properties.md
{% endcontent-ref %}
```

Transaction Fees

The sender pays for the token association fee and the rent for the first auto-renewal period.

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate the cost of your transaction fee.

Transaction Signing Requirements

The account key(s) are required to sign the transaction.

If you are updating the keys on the account, the OLD KEY and NEW KEY must sign.

If either is a key list, the key list keys are all required to sign.

If either is a threshold key, the threshold value is required to sign.

If you do not have the required signatures, the network will throw an INVALIDSIGNATURE error.

Maximum Auto-Associations and Fees

Accounts have a property, maxAutoAssociations, and the property's value determines the maximum number of automatic token associations allowed.

Property	Description
Value	<p><code>0</code></p> <p>Automatic token associations or <a data-footnote-ref="" href="#user-content-fn-1">token airdrops are not allowed, and the account must be manually associated with a token. This also applies if the value is less than or equal to <code>usedAutoAssociations</code>.</p>
Value	<p><code>-1</code></p> <p>The number of automatic token associations an account can have is unlimited. <code>-</code></p>

1

0	If the value is a positive number (number greater than 0), the number of automatic token associations an account can have is limited to that number.
---	--

{% hint style="info" %}

The sender pays the maxAutoAssociations fee and the rent for the first auto-renewal period for the association. This is in addition to the typical transfer fees. This ensures the receiver can receive tokens without association and makes it a smoother transfer process.

{% endhint %}

Reference: HIP-904

Methods

Method	Type	Requirement
setAccountId(<accountId>)	AccountId	Required
setKey(<key>)	Key	Optional
setReceiverSignatureRequired(<boolean>)	Boolean	Optional
setMaxAutomaticTokenAssociations(<amount>)	int	Optional
setAccountMemo(<memo>)	String	Optional
setAutoRenewPeriod(<duration>)	Duration	Optional
setStakedAccountId(<stakedAccountId>)	AccountId	Optional
setStakedNodeId(<stakedNodeId>)	long	Optional
setDeclineStakingReward(<declineStakingReward>)	boolean	Optional
setExpirationTime(<expirationTime>)	Instant	Disabled

{% tabs %}

{% tab title="Java" %}

java

//Create the transaction to update the key on the account

AccountUpdateTransaction transaction = new AccountUpdateTransaction()

.setAccountId(accountId)

.setKey(updateKey);

//Sign the transaction with the old key and new key, submit to a Hedera network

TransactionResponse txResponse =

transaction.freezeWith(client).sign(oldKey).sign(newKey).execute(client);

//Request the receipt of the transaction

TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status

Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//Version 2.0.0

{% endtab %}

{% tab title="JavaScript" %}

javascript

//Create the transaction to update the key on the account


```

const transaction = await new AccountUpdateTransaction()
    .setAccountId(accountId)
    .setKey(updateKey)
    .freezeWith(client);

//Sign the transaction with the old key and new key
const signTx = await (await transaction.sign(oldKey)).sign(newKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction to update the key on the account
transaction, err := hedera.NewAccountUpdateTransaction().
    SetAccountID(newAccountId).
    SetKey(updateKey.PublicKey()).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign the transaction with the old key and new key, submit to a Hedera network
txResponse, err := transaction.Sign(newKey).Sign(updateKey).Execute(client)

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

println("The transaction consensus status is ", transactionStatus)

//Version 2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

Return the properties of an account create transaction.

```

Method	Type	Description
<code>getKey()</code>	Key	Returns the public key on the account
<code>getInitialBalance()</code>	Hbar	

td>	Returns the initial balance of the account
td>	<code>getReceiverSignatureRequired()</code>
td>	boolean
td>	Returns whether the receiver signature is required or not
td>	<code>getExpirationTime()</code>
td>	Instant
td>	Returns the expiration time
td>	<code>getAccountMemo()</code>
td>	String
td>	Returns the account memo
td>	<code>getDeclineStakingReward()</code>
td>	boolean
td>	Returns whether or not the account is declining rewards
td>	<code>getStakedNodeId()</code>
td>	long
td>	Returns the node ID the account is staked to
td>	<code>getStakedAccountId()</code>
td>	AccountId
td>	Returns the account ID the node is staked to
td>	<code>getAutoRenewPeriod()</code>
td>	Duration
td>	Returns the auto renew period on the account

```
{% tabs %}
{% tab title="Java" %}
java
//Create a transaction
AccountUpdateTransaction transaction = new AccountUpdateTransaction()
    .setAccountId(accountId)
    .setKey(newKeyUpdate);
```

```
//Get the key on the account
Key accountKey = transaction.getKey();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create a transaction
const transaction = new AccountUpdateTransaction()
    .setAccountId(accountId)
    .setKey(newKeyUpdate);
```

```
//Get the key of an account
const accountKey = transaction.getKey();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
//Create the transaction
transaction, err := hedera.NewAccountUpdateTransaction().
    SetAccountId(newAccountId).
    SetKey(updateKey.PublicKey())
```

```
//Get the key of an account
accountKey := transaction.GetKey()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

```
[^1]:
```

```
# create-a-topic.md:
```

Create a topic

A transaction that creates a new topic recognized by the Hedera network. The newly generated topic can be referenced by its `topicId`. The `topicId` is used to identify a unique topic to submit messages to. You can obtain the new topic ID by requesting the receipt of the transaction. All messages within a topic are sequenced with respect to one another and are provided a unique sequence number.

Private topic

You can also create a private topic where only authorized parties can submit messages to that topic. To create a private topic you would need to set the `submitKey` property of the transaction. The `submitKey` value is then shared with the authorized parties and is required to successfully submit messages to the private topic.

Topic Properties

Field	Description

Admin Key	Access control for <code>updateTopic/deleteTopic</code> . If no <code>adminKey</code> is specified, anyone can increase the topic's <code>expirationTime</code> with <code>updateTopic</code> , but they cannot use <code>deleteTopic</code> . However, if an <code>adminKey</code> is specified, both <code>updateTopic</code> and <code>deleteTopic</code> can be used.
Submit Key	Access control for <code>submitMessage</code> . No access control will be performed specified, allowing all message submissions.
Topic Memo (100 bytes)	Store the new topic with a short publicly visible memo.
Auto Renew Account	At the topic's <code>expirationTime</code> , the optional account can be used to extend the lifetime up to a maximum of the <code>autoRenewPeriod</code> or duration/amount that all funds on the account can extend (whichever is the smaller). Currently, rent is not enforced for topics so no auto-renew payments will be made.
Auto Renew Period	<p>The initial lifetime of the topic and the amount of time to attempt to extend the topic's lifetime by automatically at the topic's <code>expirationTime</code>. Currently, rent is not enforced for topics so auto-renew payments will not be made.</p> <p>NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: <code>AUTORENEWDURATIONNOTINRANGE</code>.</p>

Transaction Signing Requirements:

If an admin key is specified, the admin key must sign the transaction.
If an admin key is not specified, the topic is immutable.
If an auto-renew account is specified, that account must also sign this transaction.

Transaction Fees

Please see the transaction and query fees table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Requirements
setAdminKey(<adminKey>)	Key	Optional
setSubmitKey(<submitKey>)	Key	Optional
setTopicMemo(<memo>)	String	Optional
setAutoRenewAccountId(<accountId>)	AccountId	Optional
setAutoRenewPeriod(<autoRenewPeriod>)	Duration	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TopicCreateTransaction transaction = new TopicCreateTransaction();

//Sign with the client operator private key and submit the transaction to a
Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the topic ID
TopicId newTopicId = receipt.topicId;

System.out.println("The new topic ID is " + newTopicId);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new TopicCreateTransaction();

//Sign with the client operator private key and submit the transaction to a
Hedera network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the topic ID
const newTopicId = receipt.topicId;

console.log("The new topic ID is " + newTopicId);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewTopicCreateTransaction()

//Sign with the client operator private key and submit the transaction to a
Hedera network
txResponse, err := transaction.Execute(client)
```

```

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
transactionReceipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the topic ID
newTopicID := transactionReceipt.TopicID

fmt.Printf("The new topic ID is %v\n", newTopicID)

//v2.0.0

{% endtab %}
{% endtabs %}

```

Get transaction values

Method	Type	Requirements
getAdminKey(<adminKey>)	Key	Optional
getSubmitKey(<submitKey>)	Key	Optional
getTopicMemo(<memo>)	String	Optional
getAutoRenewAccountId()	AccountId	Required
getAutoRenewPeriod()	Duration	Required

```

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TopicCreateTransaction transaction = new TopicCreateTransaction()
    .setAdminKey(adminKey);

//Get the admin key from the transaction
Key getKey = transaction.getAdminKey();

//V2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new TopicCreateTransaction()
    .setAdminKey(adminKey);

//Get the admin key from the transaction
const topicAdminKey = transaction.adminKey;

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewTopicCreateTransaction().
    SetAdminKey(adminKey)

getKey := transaction.GetAdminKey()

```

```
//V2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

```
# delete-a-topic.md:
```

Delete a topic

A transaction that deletes a topic from the Hedera network. Once a topic is deleted, the topic cannot be recovered to receive messages and all submitMessage calls will fail. Older messages can still be accessed, even after the topic is deleted, via the mirror node.

Transaction Signing Requirements

If the adminKey was set upon the creation of the topic, the adminKey is required to sign to successfully delete the topic

If no adminKey was set upon the creating of the topic, you cannot delete the topic and will receive an UNAUTHORIZED error

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description	Requirement
<code>setTopicId(#{x3C;topicId>}</code></code>	TopicId	The ID of the topic to delete	Required

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction
```

```
TopicDeleteTransaction transaction = new TopicDeleteTransaction()  
    .setTopicId(newTopicId);
```

```
//Sign the transaction with the admin key, sign with the client operator and  
submit the transaction to a Hedera network, get the transaction ID
```

```
TransactionResponse txResponse =  
transaction.freezeWith(client).sign(adminKey).execute(client);
```

```
//Request the receipt of the transaction
```

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```
//Get the transaction consensus status
```

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The transaction consensus status is " +transactionStatus);
```

```
//V2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Create the transaction
```

```
const transaction = await new TopicDeleteTransaction()  
    .setTopicId(newTopicId)
```

```

        .freezeWith(client);

//Sign the transaction with the admin key
const signTx = await transaction.sign(adminKey);

//Sign with the client operator private key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction and freeze the transaction to prepare for signing
transaction := hedera.NewTopicDeleteTransaction().
    SetTopicID(topicID).
    FreezeWith(client)

//Sign the transaction with the admin key, sign with the client operator and
submit the transaction to a Hedera network, get the transaction ID
txResponse, err := transaction.Sign(adminKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method                | Type    | Description                |
Requirement |
| ----- | ----- | ----- |
| getTopicId(<topicId>) | TopicId | The ID of the topic to delete | Required
|

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction

```

```
TopicDeleteTransaction transaction = new TopicDeleteTransaction()
    .setTopicId(newTopicId);
```

```
//Get topic ID
TopicId getTopicId = transaction.getTopicId();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java
//Create the transaction
const transaction = new TopicDeleteTransaction()
    .setTopicId(newTopicId);
```

```
//Get topic ID
const getTopicId = transaction.getTopicId();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
java
//Create the transaction
transaction := hedera.NewTopicDeleteTransaction().
    SetTopicID(topicID)
```

```
//Get topic ID
getTopicId := transaction.GetTopicID()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
# errors.md:
```

```
Network Response
```

```
Network response messages and their descriptions.
```

Network Response Messages	Description

-----	-----
-----	-----
-----	-----
INVALIDTOPICID	The Topic ID specified is not in the system.
TOPICDELETED	The Topic has been deleted
INVALIDTOPICEXPIRATIONTIME	The expiration time set for the topic is not valid
INVALIDTOPICADMINKEY	The adminKey associated with the topic is not correct
INVALIDTOPICSUBMITKEY	The submitKey associated with the topic is not correct

UNAUTHORIZED	An attempted operation was not authorized (ie - a deleteTopic for a topic with no adminKey)
INVALIDTOPICMESSAGE	A ConsensusService message is empty
INVALIDAUTORENEWACCOUNT	The autoRenewAccount specified is not a valid, active account.
AUTORENEWACCOUNTNOTALLOWED	An admin key was not specified on the topic, so there must not be an autorenew account.
AUTORENEWACCOUNTSIGNATUREMISSING	The autoRenewAccount didn't sign the transaction.
INVALIDCHUNKNUMBER	Chunk number must be from 1 to total (chunks) inclusive
InvalidChunkTransactionId	For every chunk, the payer account that is part of initialTransactionID must match the Payer Account of this transaction. The entire initialTransactionID should match the transactionID of the first chunk, but this is not checked or enforced by Hedera except when the chunk number is 1.
TopicExpired	The topic has expired, was not automatically renewed, and is in a 7 day grace period before the topic will be deleted unrecoverably. This error response code will not be returned until autoRenew functionality is supported by HAPI.

get-topic-info.md:

Get topic info

Topic info returns the following values for a topic. Queries do not change the state of the topic or require network consensus. The information is returned from a single node processing the query.

Topic Info Response:

Field	Description

Topic ID	The ID of the topic
Admin Key	Access control for update/delete of the topic. Null if there is no key.
Submit Key	Access control for ConsensusService.submitMessage. Null if there is no key.
Sequence Number	Current sequence number (starting at 1 for the first submitMessage) of messages on the topic.
Running Hash	SHA-384 running hash
Expiration Time	Effective consensus timestamp at (and after) which submitMessage calls will no longer succeed on the topic and the topic will expire and be marked as deleted.
Topic Memo	Short publicly visible memo about the topic. No guarantee of uniqueness.

Auto Renew Period	The lifetime of the topic and the amount of time to extend the topic's lifetime by
Auto Renew Account	Null if there is no autoRenewAccount.
Ledger ID	The ID of the network the response came from. See HIP-198.

Query Signing Requirements

The client operator private key is required to sign the query request

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

```
<table><thead><tr><th
width="330.3333333333333">Method</th><th>Type</th><th>Requirement</th></tr></thead><tbody><tr><td><code>setTopicId(&#x3C;topicId>)</code></td><td>TopicId</td><td>Required</td></tr><tr><td><code>&#x3C;TopicInfo>.adminKey</code></td><td><a href=" ../keys/generate-a-new-key-pair.md">Key</a></td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.submitKey</code></td><td><a href=" ../keys/generate-a-new-key-pair.md">Key</a></td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.topicId</code></td><td><a href=" ../specialized-types.md#topicid">TopicId</a></td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.sequenceNumber</code></td><td>long</td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.runningHash</code></td><td>ByteString</td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.memo</code></td><td>String</td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.ledgerId</code></td><td>LedgerId</td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.expirationTime</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.autoRenewAccount</code></td><td><a href=" ../specialized-types.md#accountid">AccountId</a></td><td>Optional</td></tr><tr><td><code>&#x3C;TopicInfo>.autoRenewPeriod</code></td><td>Instant</td><td>Optional</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the account info query
TopicInfoQuery query = new TopicInfoQuery()
    .setTopicId(newTopicId);

//Submit the query to a Hedera network
TopicInfo info = query.execute(client);

//Print the account key to the console
System.out.println(info);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the account info query
const query = new TopicInfoQuery()
    .setTopicId(newTopicId);

//Submit the query to a Hedera network
const info = await query.execute(client);
```

```

//Print the account key to the console
console.log(info);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
//Create the account info query
query, err := hedera.NewTopicInfoQuery().
    SetTopicID(topicID)

//Submit the query to a Hedera network
info, err := query.Execute(client)
if err != nil {
    panic(err)
}

//Print the account key to the console
println(info)

//v2.0.0

{% endtab %}
{% endtabs %}

{% tabs %}
{% tab title="Sample Output:" %}

TopicInfo{
    topicId=0.0.102736,
    topicMemo=,
    runningHash=[
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0. 0, 0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0
    ],
    sequenceNumber=0,
    expirationTime=2021-02-09T03:17:07.258292001Z,
    adminKey=null,
    submitKey=null,
    autoRenewPeriod=PT2160H,
    autoRenewAccountId=null
}

{% endtab %}
{% endtabs %}

```

get-topic-message.md:

Get topic messages

Subscribe to a topic ID's messages from a mirror node. You will receive all messages for the specified topic or within the defined start and end time.

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Methods

Method	Type	Description
--------	------	-------------

Requirement		

setTopicId(<topicId>)	TopicId	The topic ID to subscribe to
Required		
setStartTime(<startTime>)	Instant	The time to start
subscribing to a topic's messages	Optional	
setEndTime(<endTime>)	Instant	The time to stop subscribing
to a topic's messages	Optional	
setLimit(<limit>)	long	The number of messages to
return	Optional	
subscribe(<client, onNext>)	SubscriptionHandle	Client, Consumer\
<TopicMessage>	Required	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
new TopicMessageQuery()
    .setTopicId(newTopicId)
    .subscribe(client, topicMessage -> {
        System.out.println("at " + topicMessage.consensusTimestamp + " ( seq = "
+ topicMessage.sequenceNumber + " ) received topic message of " +
topicMessage.contents.length + " bytes");
    });

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
new TopicMessageQuery()
    .setTopicId(topicId)
    .setStartTime(0)
    .subscribe(
        client,
        (message) => console.log(Buffer.from(message.contents,
"utf8").toString())
    );
//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the query
, err = hedera.NewTopicMessageQuery().
    SetTopicID(topicID).
    Subscribe(client, func(message hedera.TopicMessage) {
        if string(message.Contents) == content {
            wait = false
        }
    })
if err != nil {
    panic(err)
}

//v2.0.0

{% endtab %}
{% endtabs %}
```

README.md:

Consensus Service

submit-a-message.md:

Submit a message

A transaction that submits a topic message to the Hedera network. To access the messages submitted to a topic ID, subscribe to the topic via a mirror node. The mirror node will publish the ordered messages to subscribers. Once the transaction is successfully executed, the receipt of the transaction will include the topic's updated sequence number and topic running hash.

Transaction Signing Requirements

Anyone can submit a message to a public topic

The submitKey is required to sign the transaction for a private topic

Transaction Fees

Please see the transaction and query fees table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

{% hint style="info" %}

 NOTE: Max size of an HCS message: 1024 bytes (1 kb).

{% endhint %}

Methods

Method	Type	Description	Requirement
<code>setTopicId(&#x3C;topicId>)</code>	TopicId	The topic ID to submit the message	Required
<code>setMessage(&#x3C;message>)</code>	String	The message in a String format	Optional
<code>setMessage(&#x3C;message>)</code>	byte []	The message in a byte array format	Optional
<code>setMessage(&#x3C;message>)</code>	ByteString	The message in a ByteString format	Optional

{% tabs %}

{% tab title="Java" %}

java

//Create the transaction

TopicMessageSubmitTransaction transaction = new TopicMessageSubmitTransaction()

.setTopicId(newTopicId)

.setMessage("hello, HCS! ");

//Sign with the client operator key and submit transaction to a Hedera network,
get transaction ID

TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction

TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status

Status transactionStatus = receipt.status;

```

System.out.println("The transaction consensus status is " +transactionStatus);
//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
await new TopicMessageSubmitTransaction({
    topicId: createReceipt.topicId,
    message: "Hello World",
}).execute(client);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewTopicSubmitTransaction().
    SetTopicID(topicID).
    SetMessage([]byte(content))

//Sign with the client operator private key and submit the transaction to a
Hedera network
txResponse, err := transaction.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
transactionReceipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", transactionStatus)
//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method | Type | Description |
| --- | --- | --- |
| getTopicId() | TopicId | The topic ID to submit the message to |
| getMessage() | ByteString | The message being submitted |
| getAllTransactionHash() | byte \[ ] | The hash for each transaction |

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TopicMessageSubmitTransaction transaction = new TopicMessageSubmitTransaction()
    .setTopicId(newTopicId)
    .setMessage("hello, HCS! ");

```

```

//Get the transaction message
ByteString getMessage = transaction.getMessage();
//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new TopicMessageSubmitTransaction()
    .setTopicId(newTopicId)
    .setMessage("hello, HCS! ");

//Get the transaction message
const getMessage = transaction.getMessage();

//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewTopicSubmitTransaction().
    SetTopicID(topicID).
    SetMessage([]byte(content))

//Get the transaction message
getMessage := transaction.GetMessage()
//v2.0.0

{% endtab %}
{% endtabs %}

```

update-a-topic.md:

Update a topic

A transaction that updates the properties of an existing topic. This includes the topic memo, admin key, submit key, auto-renew account, and auto renew period.

Topic Properties

Field	Description

Topic ID	Update the topic ID
Admin Key	Set a new admin key that authorizes update topic and delete topic transactions.
Submit Key	Set a new submit key for a topic that authorizes sending messages to this topic.
Topic Memo	Set a new short publicly visible memo on the new topic and is stored with the topic. (100 bytes)

|
| Auto Renew Account | Set a new auto-renew account ID for this topic.
Currently, rent is not enforced for topics so auto-renew payments will not be made.

|
| Auto Renew Period | <p>Set a new auto-renew period for this topic. Currently, rent is not enforced for topics so auto-renew payments will not be made.

NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: AUTORENEWDURATIONNOTINRANGE.</p> |

Transaction Signing Requirements

If an admin key is updated, the transaction must be signed by the pre-update admin key and post-update admin key.

If the admin key was set during the creation of the topic, the admin key must sign the transaction to update any of the topic's properties

If no adminKey was defined during the creation of the topic, you can only extend the expirationTime.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

```
<table><thead><tr><th
width="417.3333333333333">Method</th><th>Type</th><th>Requirements</th></tr></thead><tbody><tr><td><code>setTopicId(&#x3C;topicId>)</code></td><td>TopicId</td><td>Required</td></tr><tr><td><code>setAdminKey(&#x3C;adminKey>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setSubmitKey(&#x3C;submitKey>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setExpirationTime(&#x3C;expirationTime>)</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>setTopicMemo(&#x3C;memo>)</code></td><td>String</td><td>Optional</td></tr><tr><td><code>setAutoRenewAccountId(&#x3C;accountId>)</code></td><td>AccountId</td><td>Optional</td></tr><tr><td><code>setAutoRenewPeriod(&#x3C;autoRenewPeriod>)</code></td><td>Duration</td><td>Optional</td></tr><tr><td><code>clearAdminKey()</code></td><td></td><td>Optional</td></tr><tr><td><code>clearSubmitKey()</code></td><td></td><td>Optional</td></tr><tr><td><code>clearTopicMemo()</code></td><td></td><td>Optional</td></tr><tr><td><code>clearAutoRenewAccountId()</code></td><td></td><td>Optional</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
    //Create a transaction to add a submit key
    TopicUpdateTransaction transaction = new TopicUpdateTransaction()
        .setTopicId(topicId)
        .setSubmitKey(submitKey);

    //Sign the transaction with the admin key to authorize the update
    TopicUpdateTransaction signTx = transaction.freezeWith(client).sign(adminKey);

    //Sign the transaction with the client operator, submit to a Hedera network, get the transaction ID
    TransactionResponse txResponse = signTx.execute(client);
```



```

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a transaction to add a submit key
const transaction = await new TopicUpdateTransaction()
    .setTopicId(topicId)
    .setSubmitKey(submitKey)
    .freezeWith(client);

//Sign the transaction with the admin key to authorize the update
const signTx = await transaction.sign(adminKey);

//Sign with the client operator private key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewTopicUpdateTransaction().
    SetTopicId(topicId).
    SetTopicMemo("new memo")

//Sign with the client operator private key and submit the transaction to a
Hedera network
txResponse, err := transaction.FreezeWith(client).Sign(adminkey)Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", transactionStatus)

```

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

Method	Type	Requirements
getTopicId()	TopicId	Required
getAdminKey()	Key	Optional
getSubmitKey()	Key	Optional
getTopicMemo()	String	Optional
getAutoRenewAccountId()	AccountId	Required
getAutoRenewPeriod()	Duration	Required

{% tabs %}
{% tab title="Java" %}
java
//Create a transaction to add a submit key
TopicUpdateTransaction transaction = new TopicUpdateTransaction()
 .setSubmitKey(submitKey);

//Get submit key
transaction.getSubmitKey()

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a transaction to add a submit key
const transaction = new TopicUpdateTransaction()
 .setSubmitKey(submitKey);

//Get submit key
transaction.getSubmitKey()

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewTopicUpdateTransaction()
 SetSubmitKey()

transaction := transaction.GetSubmitKey()

//v2.0.0

{% endtab %}
{% endtabs %}

errors.md:

Network Response Messages

Network response messages and their descriptions.

Errors	Description
--------	-------------

ACCOUNTIDDOESNOTEXIST passed has not yet been created.	The account id
ACCOUNTUPDATEFAILED the account failed	The update of
ACCOUNTDELETED has been marked as deleted	The account
INVALIDACCOUNTAMOUNTS transfer credit and debit do not sum equal to 0	The crypto
INVALIDINITIALBALANCE the negative initial balance	Attempt to set
INVALIDRECEIVERRECORDTHRESHOLD negative receive record threshold	Attempt to set
INVALIDSENDRECORDTHRESHOLD negative send record threshold	Attempt to set
SETTINGNEGATIVEACCOUNTBALANCE set negative balance value for the crypto account	Attempting to
TRANSFERLISTSIZELIMITEXCEEDED number of accounts (both from and to) allowed for crypto transfer list	Exceeded the
TRANSFERACCOUNTSAMEASDELETEACCOUNT should not be same as Account to be deleted	Transfer Account
NOREMAININGAUTOMATICASSOCIATIONS reached the limit on the automatic associations count.	The account has
EXISTINGAUTOMATICASSOCIATIONSEXCEEDGIVENLIMIT automatic associations are more than the new maximum automatic associations.	Already existing
REQUESTEDNUMAUTOMATICASSOCIATIONSEXCEEDSASSOCIATIONLIMIT number of automatic associations for an account more than the maximum allowed token associations tokens.maxPerAccount	Cannot set the

append-to-a-file.md:

Append to a file

A transaction that appends new file content to the end of an existing file. The contents of the file can be viewed by submitting a FileContentsQuery request.

Transaction Signing Requirements

The key on the file is required to sign the transaction if different than the client operator account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Constructor	Description
-----	-----

| FileAppendTransaction() | Initializes the FileAppendTransaction object |

```
java
new FileAppendTransaction()
```

```
{% hint style="info" %}
```

The default max transaction fee (1 hbar) is not enough to create a file. Use `setMaxTransactionFee()` to change the default max transaction fee from 1 hbar to 2 hbars. The default chunk size is 2,048 bytes.

```
{% endhint %}
```

Methods

Method	Type	Description
Requirement		
-----	-----	-----

setFileId(<fileId>)	FileId	The ID of the file to append
Required		
setContents(<text>)	String	The content in String format
Optional		
setContents(<content>)	byte \[]	The content in byte format
Optional		
setChunkSize(<chunkSize>)	int	The chunk size
Optional		
setMaxChunkSize(<maxChunks>)	int	The max chunk size
Optional		

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction
```

```
FileAppendTransaction transaction = new FileAppendTransaction()
```

```
    .setFileId(newFileId)
```

```
    .setContents("The appended contents");
```

```
//Change the default max transaction fee to 2 hbars
```

```
FileCreateTransaction modifyMaxTransactionFee =
```

```
transaction.setMaxTransactionFee(new Hbar(2));
```

```
//Prepare transaction for signing, sign with the key on the file, sign with the  
client operator key and submit to a Hedera network
```

```
TransactionResponse txResponse =
```

```
modifyMaxTransactionFee.freezeWith(client).sign(key).execute(client);
```

```
//Request the receipt
```

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```
//Get the transaction consensus status
```

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The transaction consensus status is " +transactionStatus);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Create the transaction
```

```
const transaction = await new FileAppendTransaction()
```

```
    .setFileId(newFileId)
```

```
    .setContents("The appended contents")
```

```

        .setMaxTransactionFee(new Hbar(2))
        .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction2 := hedera.NewFileAppendTransaction().
    SetFileID(newFileId).
    SetContents([]byte("The appended contents"))

//Change the default max transaction fee to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare transaction for signing,
freezeTransaction, err := modifyMaxTransactionFee.FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse2 err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Println("The transaction consensus status is ", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

```

Method	Type	Description	Requirement
--------	------	-------------	-------------

	-----	-----	-----	-----
getFileId()	FileId	The file ID in the transaction	Optional	
getContents()	String	The content in the transaction	Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
FileAppendTransaction transaction = new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents");
```

```
//Get the contents
ByteString getContents = transaction.getContents();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
java
//Create the transaction
const transaction = new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents");
```

```
//Get the contents
const getContents = transaction.getContents();
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
//Create the transaction
transaction2 := hedera.NewFileAppendTransaction().
    SetFileID(newFileId).
    SetContents([]byte("The appended contents"))
```

```
//Get the contents
getContents2 := transaction2.GetContents()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

create-a-file.md:

Create a file

A transaction that creates a new file on a Hedera network. The file is referenced by its file ID which can be obtained from the receipt or record once the transaction reaches consensus on a Hedera network. The file does not have a file name. If the file is too big to create with a single `FileCreateTransaction()`, the file can be appended with the remaining content multiple times using the `FileAppendTransaction()`.

```
{% hint style="info" %}
The maximum file size is 1,024 kB.
{% endhint %}
```

Transaction Signing Requirements

The key on the file is required to sign the transaction if different than the client operator account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

File Properties

[illegible]

Methods

Constructor	Description
<code>new FileCreateTransaction()</code>	Initializes the <code>FileCreateTransaction</code> object

```
java
new FileCreateTransaction()
```

```
{% hint style="info" %}
The default max transaction fee (1 hbar) is not enough to create a file. Use
setDefaultMaxTransactionFee() to change the default max transaction fee from 1
hbar to 2 hbars.
{% endhint %}
```

Method	Type	Requirement
setKeys(<keys>)	Key	Required
setContents(<contents>)	String	Optional
setContents(<bytes>)	bytes \[]	Optional
setExpirationTime(<expirationTime>)	Instant	Optional
setFileMemo(<memo>)	String	Optional

```
{% tabs %}
{% tab title="Java" %}
```

```

java
//Create the transaction
FileCreateTransaction transaction = new FileCreateTransaction()
    .setKeys(fileKey)
    .setContents(fileContents);

//Change the default max transaction fee to 2 hbars
FileCreateTransaction modifyMaxTransactionFee =
transaction.setMaxTransactionFee(new Hbar(2));

//Prepare transaction for signing, sign with the key on the file, sign with the
client operator key and submit to a Hedera network
TransactionResponse txResponse =
modifyMaxTransactionFee.freezeWith(client).sign(fileKey).execute(client);

//Request the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the file ID
FileId newFileId = receipt.fileId;

System.out.println("The new file ID is: " + newFileId);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileCreateTransaction()
    .setKeys([filePublicKey]) //A different key then the client operator key
    .setContents("the file contents")
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator private key and submit to a Hedera network
const submitTx = await signTx.execute(client);

//Request the receipt
const receipt = await submitTx.getReceipt(client);

//Get the file ID
const newFileId = receipt.fileId;

console.log("The new file ID is: " + newFileId);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewFileCreateTransaction().
    SetKeys(filePublicKey).
    SetContents([]byte("Hello, World"))

//Change the default max transaction fee to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

```



```
//Prepare transaction for signing,
freezeTransaction, err := modifyMaxTransactionFee.FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the file ID
newFileId := receipt.FileID

fmt.Printf("The new file ID is %v\n", newFileId)
//v2.0.0

{% endtab %}
{% endtabs %}
```

Get transaction values

Method	Type	Requirement
-----	-----	-----
getKeys()	Key	Optional
getContents()	ByteString	Optional
getExpirationTime()	Instant	Optional
getFileMemo()	String	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
FileCreateTransaction transaction = new FileCreateTransaction()
    .setKeys(key)
    .setContents(fileContents);

//Get the file contents
ByteString getContents = transaction.getContents();

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new FileCreateTransaction()
    .setKeys(key)
    .setContents(fileContents);

//Get the file contents
const getContents = transaction.getContents();

{% endtab %}

{% tab title="Go" %}
```

```

java
//Create the transaction
transaction := hedera.NewFileCreateTransaction().
    SetKeys(filePublicKey).
    SetContents([]byte("Hello, World"))

```

```

//Get the file contents
getContents := transaction.GetContents()

```

```

{% endtab %}
{% endtabs %}

```

delete-a-file.md:

Delete a file

A transaction that deletes a file from a Hedera network. When deleted, a file's contents are truncated to zero length and it can no longer be updated or appended to, or its expiration time extended. When you request the contents or info of a deleted file, the network will return FILE\DELETED.

Transaction Signing Requirements

The key(s) on the file are required to sign the transaction
 If you do not sign with the key(s) on the file, you will receive an INVALID\ SIGNATURE network error

Transaction Fees

Please see the transaction and query fees table for base transaction fee
 Please use the Hedera fee estimator to estimate your transaction fee cost

Constructor	Description
new FileDeleteTransaction()	Initializes the FileDeleteTransaction object

```

java
new FileDeleteTransaction()

```

Methods

Method	Type	Description
setFileId(<fileId>)	FileId	The ID of the file to delete in x.y.z format

```

{% tabs %}
{% tab title="Java" %}

```

```

java
//Create the transaction
FileDeleteTransaction transaction = new FileDeleteTransaction()
    .setFileId(newFileId);

```

```

//Modify the default max transaction fee to from 1 to 2 hbars
FileDeleteTransaction modifyMaxTransactionFee =
transaction.setMaxTransactionFee(new Hbar(2));

```

```

//Prepare transaction for signing, sign with the key on the file, sign with the
client operator key and submit to a Hedera network
TransactionResponse txResponse =
modifyMaxTransactionFee.freezeWith(client).sign(key).execute(client);

```

```

//Request the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " + transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileDeleteTransaction()
    .setFileId(fileId)
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator private key and submit to a Hedera network
const submitTx = await signTx.execute(client);

//Request the receipt
const receipt = await submitTx.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus3.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewFileDeleteTransaction().
    SetFileID(fileId)

//Modify the default max transaction fee to from 1 to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare the transaction for signing
freezeTransaction, err := modifyMaxTransactionFee.FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)

```

```

if err != nil {
    panic(err)
}

//Get the transaction status
transactionStatus := receipt.Status

fmt.Println("The transaction consensus status is ", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method | Type | Description |
| ----- | ----- | ----- |
| getFileId(<fileId>) | FileId | The ID of the file to delete (x.z.y) |

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
FileDeleteTransaction transaction = new FileDeleteTransaction()
    .setFileId(newFileId);

//Get the file ID
FileId getFileId = transaction.getFileId();

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new FileDeleteTransaction()
    .setFileId(newFileId);

//Get the file ID
FileId getFileId = transaction.getFileId();

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewFileDeleteTransaction().
    SetFileID(fileId)

//Get the file ID
getFileId := transaction.GetFileID()

{% endtab %}
{% endtabs %}

# errors.md:

Network Response Messages

Network response messages and their descriptions.

| Network Response | Description |
|

```

FEESCHEDULEFILEPARTUPLOADED	Fee Schedule Proto File Part uploaded
FILECONTENTEMPTY	The contents of file are provided as empty.
FILEDELETED	the file has been marked as deleted
FILESYSTEMEXCEPTION	Unexpected exception thrown by file system functions
FILEUPLOADEDPROTOINVALID	Fee Schedule Proto uploaded but not valid (append or update is required)
FILEUPLOADEDPROTONOTSAVEDTODISK	Fee Schedule Proto uploaded but not valid (append or update is required)
INVALIDEXCHANGERATEFILE	Failed to update exchange rate file
INVALIDFEEFILE	Failed to update fee file
INVALIDFILEID	The file id is invalid or does not exist
INVALIDFILEWACL	File WACL keys are invalid
MAXFILESIZEEXCEEDED	File size exceeded the currently allowable limit
NOWACLKEY	WriteAccess Control Keys are not provided for the file

get-file-contents.md:

Get file contents

A query to get the contents of a file. Queries do not change the state of the file or require network consensus. The information is returned from a single node processing the query.

Query Signing Requirements

The client operator private key is required to sign the query request

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Constructor	Description
new FileContentsQuery()	Initializes a FileContentsQuery object

```
java
new FileContentsQuery()
```

Methods

Method	Type	Description
setFileId(<fileId>)	FileId	The ID of the file to get contents for (x.z.y)

```
{% tabs %}
{% tab title="Java" %}
```

```

java
//Create the query
FileContentsQuery query = new FileContentsQuery()
    .setFileId(newFileId);

//Sign with client operator private key and submit the query to a Hedera network
ByteString contents = query.execute(client);

//Change to Utf-8 encoding
String contentsToUtf8 = contents.toStringUtf8();

System.out.println(contentsToUtf8);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileContentsQuery()
    .setFileId(newFileId);

//Sign with client operator private key and submit the query to a Hedera network
const contents = await query.execute(client);

console.log(contents.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileContentsQuery().
    SetFileID(newFileId)

//Sign with client operator private key and submit the query to a Hedera network
contents, err := query.Execute(client)

fmt.Println(string(contents))

//v2.0.0

{% endtab %}
{% endtabs %}

Get query values

| Method          | Type   | Description |
| ----- | ----- | ----- |
| getFileId() | FileId | The ID of the file to get contents for (x.z.y) |

{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileContentsQuery query = new FileContentsQuery()
    .setFileId(newFileId);

//Get file ID
FileId getFileId = query.getFileId();

```

```
//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileContentsQuery()
    .setFileId(newFileId);

//Get file ID
const getFileId = query.getFileId();

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileContentsQuery().
    SetFileID(newFileId)

//Get file ID
getFileId := query.GetFileID()

//v2.0.0

{% endtab %}
{% endtabs %}
```

get-file-info.md:

Get file info

A query that returns the current state of a file. Queries do not change the state of the file or require network consensus. The information is returned from a single node processing the query.

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

File Info Response

Field	Description

File ID	The Hedera ID of the file
Key(s)	The current admin key(s) on the account
Size	The number of bytes in the file contents
Expiration Time	The current time at which the file is set to expire
Deleted	Whether or not the file has been deleted
Ledger ID	The ID of the network the response came from. See HIP-198.
Memo	A short description, if any

\

Query Signing Requirements

The client operator account paying for the query fees is required to sign

Constructor	Description
new FileInfoQuery()	Initializes the FileInfoQuery object

```
java
new FileInfoQuery()
```

Methods

Method	Type	Description
setFileId(<fileId>)	FileId	The ID of the file to get information for (x.y.z)

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileInfoQuery query = new FileInfoQuery()
    .setFileId(fileId);

//Sign the query with the client operator private key and submit to a Hedera
network
FileInfo getInfo = query.execute(client);

System.out.println("File info response: " +getInfo);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileInfoQuery()
    .setFileId(fileId);

//Sign the query with the client operator private key and submit to a Hedera
network
const getInfo = await query.execute(client);

console.log("File info response: " + getInfo);

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileInfoQuery().
    SetFileID(newFileId)

//Sign the query with the client operator private key and submit to a Hedera
network
getInfo, err := query.Execute(client)

fmt.Println(getInfo)

{% endtab %}
```



```
{% endtabs %}
```

Sample Output:

```
FileInfo{
    fileId=0.0.104926,
    size=26,
    expirationTime=2021-02-10T17:48:15Z,
    deleted=false,

    keys=[ 302a300506032b6570032100100059296cc51f5d362a3859d3c3c74c6a480cffad9d669a1
0c1d447ce56e5bf
    ]
}
```

Get query values

Method	Type	Description
getFileId()	FileId	The ID of the file to get contents for (x.z.y)

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileInfoQuery query = new FileInfoQuery()
    .setFileId(fileId);
```

```
//Get file ID
FileId getFileId = query.getFileId();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileInfoQuery()
    .setFileId(fileId);
```

```
//Get file ID
const getFileId = query.getFileId();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileContentsQuery().
    SetFileID(newFileId)
```

```
//Get file ID
getFileId := query.GetFileID()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

README.md:

File Service

update-a-file.md:

Update a file

A transaction that updates the state of an existing file on a Hedera network. Once the transaction has been processed, the network will be updated with the new field values of the file. If you need to access a previous state of the file, you can query a mirror node.

Transaction Signing Requirements

The key or keys on the file are required to sign this transaction to modify the file properties

If you are updating the keys on the file, you must sign with the old key and the new key

If you do not sign with the key(s) on the file, you will receive an INVALIDSIGNATURE network error

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

File Properties

Field	Description

-----	-----
-----	-----
-----	-----
-----	-----
--	
Key(s)	Update the keys which must sign any transactions modifying this file. All keys must sign to modify the file's contents or keys. No key is required to sign for extending the expiration time (except the one for the operator account paying for the transaction). The network currently requires a file to have at least one key (or key list or threshold key) but this requirement may be lifted in the future.
Contents	The content to update the files with.
Expiration Time	If set, update the expiration time of the file. Must be in the future (may only be used to extend the expiration). To make a file inaccessible use FileDeleteTransaction.
Memo	Short publicly visible memo about the file. No guarantee of uniqueness. (100 characters max)
Constructor	Description
-----	-----
new FileUpdateTransaction()	Initializes the FileUpdateTransaction object

```
java
new FileUpdateTransaction()
```

Methods

{% hint style="info" %}

Note: The total size for a given transaction is limited to 6KiB. If you exceed this value you will need to submit a FileUpdateTransaction that is less than 6KiB and then submit a FileAppendTransaction to add the remaining content to the file.

{% endhint %}

Method	Type	Requirement
setFileId(<fileId>)	FileId	Required
setKey(<keys>)	Key	Optional
setContents(<bytes>)	byte \[]	Optional
setContents(<text>)	String	Optional
setExpirationTime(<expiration>)	Instant	Optional
setFileMemo(<memo>)	String	Optional

{% tabs %}

{% tab title="Java" %}

java

//Create the transaction

```
FileUpdateTransaction transaction = new FileUpdateTransaction()
    .setFileId(fileId)
    .setKeys(newKey);
```

//Modify the max transaction fee

```
FileUpdateTransaction txFee = transaction.setMaxTransactionFee(new Hbar(3));
```

//Freeze the transaction, sign with the original key, sign with the new key, sign with the client operator key and submit the transaction to a Hedera network

```
TransactionResponse txResponse =
txFee.freezeWith(client).sign(fileKey).sign(newKey).execute(client);
```

//Get the receipt of the transaction

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

//Get the transaction consensus status

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The transaction consensus status is " +transactionStatus);
```

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}

javascript

//Create the transaction

```
const transaction = await new FileUpdateTransaction()
    .setFileId(fileId)
    .setContents("The new contents")
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);
```

//Sign with the file private key

```
const signTx = await transaction.sign(fileKey);
```

//Sign with the client operator private key and submit to a Hedera network

```
const submitTx = await signTx.execute(client);
```

//Request the receipt

```
const receipt = await submitTx.getReceipt(client);
```

```

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus3.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewFileUpdateTransaction().
    SetFileID(fileId).
    SetKeys(newKey)

//Modify the max transaction fee
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare the transaction for signing
freezeTransaction, err := modifyMaxTransactionFee FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Sign(newKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction status
transactionStatus := receipt.Status

fmt.Println("The transaction consensus status is ", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method | Type | Requirement |
| ----- | ----- | ----- |
| getFileId() | FileId | Optional |
| getKey() | Key | Optional |
| setContents() | ByteString | Optional |
| getExpirationTime() | Instant | Optional |
| getFileMemo() | String | Optional |

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction

```

```
FileUpdateTransaction transaction = new FileUpdateTransaction()
    .setFileId(fileId)
    .setKeys(newKey);
```

```
//Get the contents of a file
Key getKey = transaction.getKey();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java
//Create the transaction
const transaction = new FileUpdateTransaction()
    .setFileId(newFileId);
```

```
//Get the contents of a file
const getKey = transaction.getKey();
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
java
//Create the transaction
transaction := hedera.NewFileUpdateTransaction().
    SetFileID(fileId).
    SetKeys(newKey)
```

```
//Get the contents of a file
getKey := transaction.GetKeys()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

append-to-a-file.md:

Append to a file

A transaction that appends new file content to the end of an existing file. The contents of the file can be viewed by submitting a FileContentsQuery request.

Transaction Signing Requirements

The key on the file is required to sign the transaction if different than the client operator account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Constructor	Description
FileAppendTransaction()	Initializes the FileAppendTransaction object

```
java
new FileAppendTransaction()
```

```
{% hint style="info" %}
```

The default max transaction fee (1 hbar) is not enough to create a file. Use `setMaxTransactionFee()` to change the default max transaction fee from 1 hbar to 2 hbars. The default chunk size is 2,048 bytes.

```
{% endhint %}
```

Methods

Method	Type	Description
Requirement		
-----	-----	-----
<code>setFileId(<fileId>)</code>	<code>FileId</code>	The ID of the file to append
Required		
<code>setContents(<text>)</code>	<code>String</code>	The content in String format
Optional		
<code>setContents(<content>)</code>	<code>byte []</code>	The content in byte format
Optional		
<code>setChunkSize(<chunkSize>)</code>	<code>int</code>	The chunk size
Optional		
<code>setMaxChunkSize(<maxChunks>)</code>	<code>int</code>	The max chunk size
Optional		

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
FileAppendTransaction transaction = new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents");

//Change the default max transaction fee to 2 hbars
FileCreateTransaction modifyMaxTransactionFee =
transaction.setMaxTransactionFee(new Hbar(2));

//Prepare transaction for signing, sign with the key on the file, sign with the
client operator key and submit to a Hedera network
TransactionResponse txResponse =
modifyMaxTransactionFee.freezeWith(client).sign(key).execute(client);

//Request the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents")
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator key and submit to a Hedera network
```

```

const txResponse = await signTx.execute(client);

//Request the receipt
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction2 := hedera.NewFileAppendTransaction().
    SetFileID(newFileId).
    SetContents([]byte("The appended contents"))

//Change the default max transaction fee to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare transaction for signing,
freezeTransaction, err := modifyMaxTransactionFee FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse2 err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Println("The transaction consensus status is ", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

| Method          | Type   | Description                                     | Requirement |
| ----- | ----- | ----- | ----- |
| getFileId()     | FileId | The file ID in the transaction | Optional    |
| getContents()   | String | The content in the transaction | Optional    |

{% tabs %}
{% tab title="Java" %}
java

```

```
//Create the transaction
FileAppendTransaction transaction = new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents");
```

```
//Get the contents
ByteString getContents = transaction.getContents();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java
```

```
//Create the transaction
```

```
const transaction = new FileAppendTransaction()
    .setFileId(newFileId)
    .setContents("The appended contents");
```

```
//Get the contents
```

```
const getContents = transaction.getContents();
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
java
```

```
//Create the transaction
```

```
transaction2 := hedera.NewFileAppendTransaction().
    SetFileID(newFileId).
    SetContents([]byte("The appended contents"))
```

```
//Get the contents
```

```
getContents2 := transaction2.GetContents()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
# create-a-file.md:
```

```
Create a file
```

A transaction that creates a new file on a Hedera network. The file is referenced by its file ID which can be obtained from the receipt or record once the transaction reaches consensus on a Hedera network. The file does not have a file name. If the file is too big to create with a single `FileCreateTransaction()`, the file can be appended with the remaining content multiple times using the `FileAppendTransaction()`.

```
{% hint style="info" %}
```

```
The maximum file size is 1,024 kB.
```

```
{% endhint %}
```

Transaction Signing Requirements

The key on the file is required to sign the transaction if different than the client operator account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

File Properties

Field	Description
-----	-----
-----	-----
-----	-----
-----	-----
-----	-----
Key(s)	Set the keys which must sign any transactions modifying this file (the owner(s) of the file). All keys must sign to modify the file's contents or keys. No key is required to sign for extending the expiration time (except the one for the operator account paying for the transaction). The network currently requires a file to have at least one key (or key list or threshold key) but this requirement may be lifted in the future.
Contents	The contents of the file. The file contents can be recovered from requesting the FileContentsQuery. Note that the total size for a given transaction is limited to 6KiB (as of March 2020) by the network; if you exceed this you may receive a TRANSACTION\OVERSIZE error.
Expiration Time	Set the instant at which this file will expire, after which its contents will no longer be available. Defaults to 1/4 of a Julian year from the instant was invoked.
Memo	Short publicly visible memo about the file. No guarantee of uniqueness. (100 characters max)

Methods

Constructor	Description
-----	-----
new FileCreateTransaction()	Initializes the FileCreateTransaction object

java

```
new FileCreateTransaction()
```

```
{% hint style="info" %}
```

The default max transaction fee (1 hbar) is not enough to create a a file. Use `setDefaultMaxTransactionFee()` to change the default max transaction fee from 1 hbar to 2 hbars.

```
{% endhint %}
```

Method	Type	Requirement
-----	-----	-----
setKeys(<keys>)	Key	Required
setContents(<contents>)	String	Optional
setContents(<bytes>)	bytes \[]	Optional
setExpirationTime(<expirationTime>)	Instant	Optional
setFileMemo(<memo>)	String	Optional

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction
```

```
FileCreateTransaction transaction = new FileCreateTransaction()
```

```
    .setKeys(fileKey)
```

```
    .setContents(fileContents);
```

```
//Change the default max transaction fee to 2 hbars
```

```

FileCreateTransaction modifyMaxTransactionFee =
transaction.setMaxTransactionFee(new Hbar(2));

//Prepare transaction for signing, sign with the key on the file, sign with the
client operator key and submit to a Hedera network
TransactionResponse txResponse =
modifyMaxTransactionFee.freezeWith(client).sign(fileKey).execute(client);

//Request the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the file ID
FileId newFileId = receipt.fileId;

System.out.println("The new file ID is: " + newFileId);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileCreateTransaction()
    .setKeys([filePublicKey]) //A different key then the client operator key
    .setContents("the file contents")
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator private key and submit to a Hedera network
const submitTx = await signTx.execute(client);

//Request the receipt
const receipt = await submitTx.getReceipt(client);

//Get the file ID
const newFileId = receipt.fileId;

console.log("The new file ID is: " + newFileId);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewFileCreateTransaction().
    SetKeys(filePublicKey).
    SetContents([]byte("Hello, World"))

//Change the default max transaction fee to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare transaction for signing,
freezeTransaction, err := modifyMaxTransactionFee FreezeWith(client)
if err != nil {
    panic(err)
}

```

```
//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}
```

```
//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}
```

```
//Get the file ID
newFileId := receipt.FileID
```

```
fmt.Printf("The new file ID is %v\n", newFileId)
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

Get transaction values

Method	Type	Requirement
-----	-----	-----
getKeys()	Key	Optional
getContents()	ByteString	Optional
getExpirationTime()	Instant	Optional
getFileMemo()	String	Optional

```
{% tabs %}
{% tab title="Java" %}
```

```
java
//Create the transaction
FileCreateTransaction transaction = new FileCreateTransaction()
    .setKeys(key)
    .setContents(fileContents);
```

```
//Get the file contents
ByteString getContents = transaction.getContents();
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
```

```
//Create the transaction
const transaction = new FileCreateTransaction()
    .setKeys(key)
    .setContents(fileContents);
```

```
//Get the file contents
const getContents = transaction.getContents();
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
```

```
//Create the transaction
transaction := hedera.NewFileCreateTransaction().
    SetKeys(filePublicKey).
    SetContents([]byte("Hello, World"))
```

```
//Get the file contents
```

```
getContents := transaction.GetContents()
```

```
{% endtab %}  
{% endtabs %}
```

```
# delete-a-file.md:
```

Delete a file

A transaction that deletes a file from a Hedera network. When deleted, a file's contents are truncated to zero length and it can no longer be updated or appended to, or its expiration time extended. When you request the contents or info of a deleted file, the network will return FILE\DELETED.

Transaction Signing Requirements

The key(s) on the file are required to sign the transaction
If you do not sign with the key(s) on the file, you will receive an INVALID\ SIGNATURE network error

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Constructor	Description
new FileDeleteTransaction()	Initializes the FileDeleteTransaction object

```
java  
new FileDeleteTransaction()
```

Methods

Method	Type	Description
setFileId(<fileId>)	FileId	The ID of the file to delete in x.y.z format

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Create the transaction  
FileDeleteTransaction transaction = new FileDeleteTransaction()  
    .setFileId(newFileId);
```

```
//Modify the default max transaction fee to from 1 to 2 hbars  
FileDeleteTransaction modifyMaxTransactionFee =  
transaction.setMaxTransactionFee(new Hbar(2));
```

```
//Prepare transaction for signing, sign with the key on the file, sign with the  
client operator key and submit to a Hedera network  
TransactionResponse txResponse =  
modifyMaxTransactionFee.freezeWith(client).sign(key).execute(client);
```

```
//Request the receipt  
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```
//Get the transaction consensus status  
Status transactionStatus = receipt.status;
```

```

System.out.println("The transaction consensus status is " + transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileDeleteTransaction()
    .setFileId(fileId)
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator private key and submit to a Hedera network
const submitTx = await signTx.execute(client);

//Request the receipt
const receipt = await submitTx.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus3.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewFileDeleteTransaction().
    SetFileID(fileId)

//Modify the default max transaction fee to from 1 to 2 hbars
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare the transaction for signing
freezeTransaction, err := modifyMaxTransactionFee FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction status
transactionStatus := receipt.Status

```

```
fmt.Println("The transaction consensus status is ", transactionStatus)
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

Get transaction values

Method	Type	Description
getFileId(<fileId>)	FileId	The ID of the file to delete (x.z.y)

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Create the transaction  
FileDeleteTransaction transaction = new FileDeleteTransaction()  
    .setFileId(newFileId);
```

```
//Get the file ID  
FileId getFileId = transaction.getFileId();
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}  
javascript  
//Create the transaction  
const transaction = new FileDeleteTransaction()  
    .setFileId(newFileId);
```

```
//Get the file ID  
FileId getFileId = transaction.getFileId();
```

```
{% endtab %}
```

```
{% tab title="Go" %}  
java  
//Create the transaction  
transaction := hedera.NewFileDeleteTransaction().  
    SetFileID(fileId)
```

```
//Get the file ID  
getFileId := transaction.GetFileID()
```

```
{% endtab %}  
{% endtabs %}
```

get-file-contents.md:

Get file contents

A query to get the contents of a file. Queries do not change the state of the file or require network consensus. The information is returned from a single node processing the query.

Query Signing Requirements

The client operator private key is required to sign the query request

Query Fees

Please see the transaction and query fees table for the base transaction fee

Please use the Hedera fee estimator to estimate your query fee cost

Constructor	Description
new FileContentsQuery()	Initializes a FileContentsQuery object

```
java
new FileContentsQuery()
```

Methods

Method	Type	Description
setFileId(<fileId>)	FileId	The ID of the file to get contents for (x.z.y)

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileContentsQuery query = new FileContentsQuery()
    .setFileId(newFileId);

//Sign with client operator private key and submit the query to a Hedera network
ByteString contents = query.execute(client);

//Change to Utf-8 encoding
String contentsToUtf8 = contents.toStringUtf8();

System.out.println(contentsToUtf8);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileContentsQuery()
    .setFileId(newFileId);

//Sign with client operator private key and submit the query to a Hedera network
const contents = await query.execute(client);

console.log(contents.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileContentsQuery().
    SetFileID(newFileId)

//Sign with client operator private key and submit the query to a Hedera network
contents, err := query.Execute(client)

fmt.Println(string(contents))
```

//v2.0.0

```
{% endtab %}  
{% endtabs %}
```

Get query values

Method	Type	Description
getFileId()	FileId	The ID of the file to get contents for (x.z.y)

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Create the query  
FileContentsQuery query = new FileContentsQuery()  
    .setFileId(newFileId);
```

```
//Get file ID  
FileId getFileId = query.getFileId();
```

//v2.0.0

```
{% endtab %}
```

```
{% tab title="JavaScript" %}  
javascript  
//Create the query  
const query = new FileContentsQuery()  
    .setFileId(newFileId);
```

```
//Get file ID  
const getFileId = query.getFileId();
```

```
{% endtab %}
```

```
{% tab title="Go" %}  
java  
//Create the query  
query := hedera.NewFileContentsQuery().  
    SetFileID(newFileId)
```

```
//Get file ID  
getFileId := query.GetFileID()
```

//v2.0.0

```
{% endtab %}  
{% endtabs %}
```

get-file-info.md:

Get file info

A query that returns the current state of a file. Queries do not change the state of the file or require network consensus. The information is returned from a single node processing the query.

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

File Info Response

Field	Description

File ID	The Hedera ID of the file
Key(s)	The current admin key(s) on the account
Size	The number of bytes in the file contents
Expiration Time	The current time at which the file is set to expire
Deleted	Whether or not the file has been deleted
Ledger ID	The ID of the network the response came from. See HIP-198.
Memo	A short description, if any

\

Query Signing Requirements

The client operator account paying for the query fees is required to sign

Constructor	Description

new FileInfoQuery()	Initializes the FileInfoQuery object

```
java
new FileInfoQuery()
```

Methods

Method	Type	Description

setFileId(<fileId>)	FileId	The ID of the file to get information for
(x.y.z)		

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileInfoQuery query = new FileInfoQuery()
    .setFileId(fileId);

//Sign the query with the client operator private key and submit to a Hedera
network
FileInfo getInfo = query.execute(client);

System.out.println("File info response: " +getInfo);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileInfoQuery()
    .setFileId(fileId);
```

```
//Sign the query with the client operator private key and submit to a Hedera
network
const getInfo = await query.execute(client);

console.log("File info response: " + getInfo);

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileInfoQuery().
    SetFileID(newFileId)

//Sign the query with the client operator private key and submit to a Hedera
network
getInfo, err := query.Execute(client)

fmt.Println(getInfo)

{% endtab %}
{% endtabs %}
```

Sample Output:

```
FileInfo{
    fileId=0.0.104926,
    size=26,
    expirationTime=2021-02-10T17:48:15Z,
    deleted=false,

    keys=[ 302a300506032b6570032100100059296cc51f5d362a3859d3c3c74c6a480cffad9d669a1
0c1d447ce56e5bf
    ]
}
```

Get query values

Method	Type	Description
getFileId()	FileId	The ID of the file to get contents for (x.z.y)

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
FileInfoQuery query = new FileInfoQuery()
    .setFileId(fileId);

//Get file ID
FileId getFileId = query.getFileId();

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new FileInfoQuery()
    .setFileId(fileId);
```

```
//Get file ID
const getFileId = query.getFileId();

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewFileContentsQuery().
    SetFileID(newFileId)

//Get file ID
getFileId := query.GetFileID()

//v2.0.0

{% endtab %}
{% endtabs %}
```

README.md:

File Service

update-a-file.md:

Update a file

A transaction that updates the state of an existing file on a Hedera network. Once the transaction has been processed, the network will be updated with the new field values of the file. If you need to access a previous state of the file, you can query a mirror node.

Transaction Signing Requirements

The key or keys on the file are required to sign this transaction to modify the file properties

If you are updating the keys on the file, you must sign with the old key and the new key

If you do not sign with the key(s) on the file, you will receive an INVALID\ SIGNATURE network error

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

File Properties

Field	Description

-- |

Key(s)	Update the keys which must sign any transactions modifying this file. All keys must sign to modify the file's contents or keys. No key is
--------	---

required to sign for extending the expiration time (except the one for the operator account paying for the transaction). The network currently requires a file to have at least one key (or key list or threshold key) but this requirement may be lifted in the future. |

| Contents | The content to update the files with.

| Expiration Time | If set, update the expiration time of the file. Must be in the future (may only be used to extend the expiration). To make a file inaccessible use FileDeleteTransaction.

| Memo | Short publicly visible memo about the file. No guarantee of uniqueness. (100 characters max)

Constructor	Description
new FileUpdateTransaction()	Initializes the FileUpdateTransaction object

```
java
new FileUpdateTransaction()
```

Methods

```
{% hint style="info" %}
```

Note: The total size for a given transaction is limited to 6KiB. If you exceed this value you will need to submit a FileUpdateTransaction that is less than 6KiB and then submit a FileAppendTransaction to add the remaining content to the file.

```
{% endhint %}
```

Method	Type	Requirement
setFileId(<fileId>)	FileId	Required
setKey(<keys>)	Key	Optional
setContents(<bytes>)	byte \[]	Optional
setContents(<text>)	String	Optional
setExpirationTime(<expiration>)	Instant	Optional
setFileMemo(<memo>)	String	Optional

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction
```

```
FileUpdateTransaction transaction = new FileUpdateTransaction()
```

```
    .setFileId(fileId)
```

```
    .setKeys(newKey);
```

```
//Modify the max transaction fee
```

```
FileUpdateTransaction txFee = transaction.setMaxTransactionFee(new Hbar(3));
```

```
//Freeze the transaction, sign with the original key, sign with the new key,
sign with the client operator key and submit the transaction to a Hedera network
```

```
TransactionResponse txResponse =
```

```
txFee.freezeWith(client).sign(fileKey).sign(newKey).execute(client);
```

```
//Get the receipt of the transaction
```

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```
//Get the transaction consensus status
```

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The transaction consensus status is " +transactionStatus);
```

```

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new FileUpdateTransaction()
    .setFileId(fileId)
    .setContents("The new contents")
    .setMaxTransactionFee(new Hbar(2))
    .freezeWith(client);

//Sign with the file private key
const signTx = await transaction.sign(fileKey);

//Sign with the client operator private key and submit to a Hedera network
const submitTx = await signTx.execute(client);

//Request the receipt
const receipt = await submitTx.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus3.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewFileUpdateTransaction().
    SetFileID(fileId).
    SetKeys(newKey)

//Modify the max transaction fee
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(2,
hedera.HbarUnits.Hbar))

//Prepare the transaction for signing
freezeTransaction, err := modifyMaxTransactionFee.FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the key on the file, sign with the client operator key and submit to
a Hedera network
txResponse, err := freezeTransaction.Sign(fileKey).Sign(newKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction status
transactionStatus := receipt.Status

```

```
fmt.Println("The transaction consensus status is ", transactionStatus)
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

Get transaction values

Method	Type	Requirement
-----	-----	-----
getFileId()	FileId	Optional
getKey()	Key	Optional
setContents()	ByteString	Optional
getExpirationTime()	Instant	Optional
getFileMemo()	String	Optional

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Create the transaction  
FileUpdateTransaction transaction = new FileUpdateTransaction()  
    .setFileId(fileId)  
    .setKeys(newKey);
```

```
//Get the contents of a file  
Key getKey = transaction.getKey();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}  
java  
//Create the transaction  
const transaction = new FileUpdateTransaction()  
    .setFileId(newFileId);
```

```
//Get the contents of a file  
const getKey = transaction.getKey();
```

```
{% endtab %}
```

```
{% tab title="Go" %}  
java  
//Create the transaction  
transaction := hedera.NewFileUpdateTransaction().  
    SetFileID(fileId).  
    SetKeys(newKey)
```

```
//Get the contents of a file  
getKey := transaction.GetKeys()
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

create-a-key-list.md:

Create a key list

Create a key list key structure where all the keys in the list are required to

sign transactions that modify accounts, topics, tokens, smart contracts, or files. A key list can contain a Ed25519 or ECDSA (secp256k1)\ key type.

If all the keys in the key list key structure do not sign, the transaction will fail and return an "INVALID\SIGNATURE" error. A key list can have repeated keys. A signature for the repeated key will count as many times as the key is listed in the key list. For example, a key list has three keys. Two of the three public keys in the list are the same. When a user signs a transaction with the repeated key it will account for two out of the three keys required signature.

```
<table data-header-hidden><thead><tr><th></th><th></th><th></th><th></th><th></th><th></th></tr></thead><tbody><tr><td><strong>Method</strong></td><td><strong>Type</strong></td><td><strong>Description</strong></td></tr><tr><td><code>KeyList.of(&#x3C;keys>)</code></td><td>Key</td><td>Keys to add to the key list</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
//Generate 3 keys
PrivateKey key1 = PrivateKey.generate();
PublicKey publicKey1 = key1.getPublicKey();

PrivateKey key2 = PrivateKey.generate();
PublicKey publicKey2 = key2.getPublicKey();

PrivateKey key3 = PrivateKey.generate();
PublicKey publicKey3 = key3.getPublicKey();

//Create a key list where all 3 keys are required to sign
KeyList keyStructure = KeyList.of(key1, key2, key3);

System.println(keyStructure)

//v2.0.0
```

Sample Output

```
KeyList{threshold=null,
keys=[302e020100300506032b6570042204201cd556de918842179791d9edd75cdd2b5d34c5c73b0239ec0b34c67eedc020fd,
302e020100300506032b6570042204209ca1ce4463b71c72bba0219c37e18347a5145a9797c6546a6c99e50255c54be3,
302e020100300506032b657004220420982bb43f4947e8376e2f0ebfde086d24323b04d731da29446e5bc399ffbe06e1]
}
```

```
{% endtab %}

{% tab title="JavaScript" %}
java
//Generate 3 keys
const key1 = PrivateKey.generate();
const publicKey1 = key1.publicKey;

const key2 = PrivateKey.generate();
const publicKey2 = key2.publicKey;

const key3 = PrivateKey.generate();
const publicKey3 = key3.publicKey;
```

```

//Create a list of the keys
const publicKeyList = [];

publicKeyList.push(publicKey1);
publicKeyList.push(publicKey2);
publicKeyList.push(publicKey3);

//Create a key list where all 3 keys are required to sign
const keys = new KeyList(publicKeyList);
//v2.0.13

{% endtab %}

{% tab title="Go" %}
java
//Generate 3 keys
key1, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey1, err := key1.PublicKey()

key2, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey2, err := key2.PublicKey()

key3, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey3, err := key3.PublicKey()

//Create a key list where all 3 keys are required to sign
keys := make([]hedera.PublicKey, 3)

keys[0] = publicKey1
keys[1] = publicKey2
keys[2] = publicKey3

keyStructure := hedera.NewKeyList().AddAllPublicKeys(keys)

fmt.Printf("The key list is %v\n", keyStructure)

//v2.0.0

{% endtab %}
{% endtabs %}

```

create-a-threshold-key.md:

Create a threshold key

Create a key structure that requires the defined threshold value to sign. A threshold key can contain a Ed25519 or ECDSA (secp256k1)\ key type. You can use either the public key or the private key to create the key structure. If the

threshold requirement is not met when signing transactions, the network will return an "INVALID\SIGNATURE" error.

Method	Type	Description
KeyList.withThreshold(<thresholdValue>)	int	The number of keys required to sign transactions to modify the account i.e. transfers, update, etc

```
{% tabs %}
{% tab title="Java" %}
java
//Generate 3 keys
PrivateKey key1 = PrivateKey.generate();
PublicKey publicKey1 = key1.getPublicKey();

PrivateKey key2 = PrivateKey.generate();
PublicKey publicKey2 = key2.getPublicKey();

PrivateKey key3 = PrivateKey.generate();
PublicKey publicKey3 = key3.getPublicKey();

PrivateKey[] keys = new PrivateKey[3]; //You can also use the 3 public keys here

keys[0] = key1;
keys[1] = key2;
keys[2] = key3;

//A key structure that requires one of the 3 keys to sign
KeyList thresholdKey = KeyList.withThreshold(1);

//Add the three keys to the thresholdKey
Collections.addAll(thresholdKey, keys);

System.out.println("The 1/3 threshold key structure" +thresholdKey);

//v2.0.0

Sample Output:

KeyList{threshold=1,
keys=[

302e020100300506032b657004220420984bd6b4e0cac783654f30c8797655953c6ab432e78bc09a
34fbda594c6395ed,

302e020100300506032b657004220420a4a7bd506f33868416d53eff55b3e8a254e17accf6cb37f4
4975792ededac120,

302e020100300506032b657004220420f8a6f2ba3174391e619a87506fb0b86c6e481809563a797f
4f84715d1a471695]
}

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Generate our key lists
```

```

const privateKeyList = [];
const publicKeyList = [];
for (let i = 0; i < 4; i += 1) {
    const privateKey = PrivateKey.generate();
    const publicKey = privateKey.publicKey;
    privateKeyList.push(privateKey);
    publicKeyList.push(publicKey);
    console.log(`${i}: pub key:${publicKey}`);
    console.log(`${i}: priv key:${privateKey}`);
}

// Create our threshold key
const thresholdKey = new KeyList(publicKeyList,1);

console.log("The 1/3 threshold key structure" +thresholdKey);

//2.0.2

{% endtab %}

{% tab title="Go" %}
go
//Generate 3 keys
key1, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey1 := key1.PublicKey()

key2, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey2:= key2.PublicKey()

key3, err := hedera.GeneratePrivateKey()

if err != nil {
    panic(err)
}

publicKey3 := key3.PublicKey()

//Create a key list where all 3 keys are required to sign
keys := make([]hedera.PublicKey, 3)

keys[0] = publicKey1
keys[1] = publicKey2
keys[2] = publicKey3

//A key structure that requires one of the 3 keys to sign
thresholdKey := hedera.KeyListWithThreshold(1).
    AddAllPublicKeys(keys)

fmt.Printf("The 1/3 threshold key structure %v\n", thresholdKey)

//v2.0.0

{% endtab %}

```

```
{% endtabs %}

# generate-a-mnemonic-phrase.md:

Generate a mnemonic phrase

Generate a 12 or 24-word mnemonic phrase that can be used to recover the private
keys that are associated with it.

<table data-header-hidden><thead><tr><th></th><th
width="117.33333333333331"></th><th></th></tr></thead><tbody><tr><td><strong>Met
hod</strong></td><td><strong>Type</strong></td><td><strong>Description</
strong></td></tr><tr><td><code>Mnemonic.generate24()</code></td><td>Mnemonic</
td><td>Generates a 24-word recovery phrase that can be used to recover a private
key</td></tr><tr><td><code>Mnemonic.generate12()</code></td><td>Mnemonic</
td><td>Generates a 12-word recovery phrase that can be used to recover a private
key</td></tr></tbody></table>

{% tabs %}
{% tab title="Java" %}
java
// 24-word recovery phrase
Mnemonic mnemonic = Mnemonic.generate24();
System.out.println("mnemonic 24 word = " + mnemonic);

//12 word recovery phrase
Mnemonic mnemonic12 = Mnemonic.generate12();
System.out.println("mnemonic 12 word = " + mnemonic12);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
// generate a 24-word mnemonic
const mnemonic = await Mnemonic.generate();

console.log(mnemonic)

{% endtab %}

{% tab title="Go" %}
java
//Generate 24 word mnemonic
mnemonic24, err := hedera.GenerateMnemonic()

if err != nil {
    panic(err)
}

privateKey, err := mnemonic24.ToPrivateKey( / passphrase / "")

if err != nil {
    panic(err)
}

publicKey := privateKey.PublicKey()

fmt.Printf("mnemonic = %v\n", mnemonic)

//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

```
# generate-a-new-key-pair.md:
```

```
Generate a new key pair
```

```
ED25519
```

Create a new ED25519 key pair used to sign transactions and queries on the Hedera network. The private key is kept confidential and is used to sign transactions that modify the state of an account, topic, token, smart contract, or file entity on the network. The public key can be shared with other users on the network.

Method	Type
Description	
-----	-----
PrivateKey.generateED25519()	
PrivateKey Generates an Ed25519 private key	
PrivateKey.generateED25519().getPublicKey()	
PublicKey Derive a public key from this Ed25519 private key	
PrivateKey.generateED25519().publicKey()	
PublicKey Derive a public key from this Ed25519 private key	
PrivateKey.generateED25519().publicKey().toAccountId(<shard>,<realm>)	long,
long Construct an alias account ID from a alias public key address	
[DEPRECATED]PrivateKey.generate()	PrivateKey
Generates an Ed25519 private key	

```
{% tabs %}
{% tab title="Java" %}
```

```
java
PrivateKey privateKey = PrivateKey.generateED25519();
PublicKey publicKey = privateKey.getPublicKey();
```

```
System.out.println("private key = " + privateKey);
System.out.println("public key = " + publicKey);
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
const privateKey = await PrivateKey.generateED25519Async();
const publicKey = privateKey.publicKey;
```

```
console.log("private key = " + privateKey);
console.log("public key = " + publicKey);
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
privateKey, err := hedera.GenerateEd25519PrivateKey()
if err != nil {
    panic(err)
}
```

```

publicKey := privateKey.PublicKey()

fmt.Printf("private key = %v\n", privateKey)
fmt.Printf("public key = %v\n", publicKey)

```

```

{% endtab %}
{% endtabs %}

```

Sample Output:

```

private key =
302e020100300506032b657004220420b9c3ebac81a72aafa5490cc78111643d016d311e60869436
fbb91c73307ed35a
public key =
302a300506032b65700321001a5a62bb9f35990d3fea1a5bb7ef6f1df0a297697adef1e04510c9d4
ecc5db3f

```

ECDSA (secp256k1\)

Create a new ECDSA (secp256k1) key pair used to sign transactions and queries on a Hedera network. The private key is kept confidential and is used to sign transactions that modify the state of an account, topic, token, smart contract, or file entity on the network. The public key can be shared with other users on the network.

Method	Type
Description	
-----	-----
PrivateKey.generateECDSA()	
PrivateKey Generates an ECDSA private key	
PrivateKey.generateECDSA().getPublicKey()	
PublicKey Derive a public key from this ECDSA private key	
PrivateKey.generateECDSA().publicKey()	
PublicKey Derive a public key from this ECDSA private key	
PrivateKey.generateECDSA().publicKey().toAccountId(<shard>, <realm>)	long,
long Constructs an account ID from an account alias public key	

```

{% tabs %}
{% tab title="Java" %}

```

```

java
PrivateKey privateKey = PrivateKey.generateECDSA();
PublicKey publicKey = privateKey.getPublicKey();

System.out.println("private key = " + privateKey);
System.out.println("public key = " + publicKey);

```

```

{% endtab %}

```

```

{% tab title="JavaScript" %}

```

```

javascript
const privateKey = PrivateKey.generateECDSA();
const publicKey = privateKey.publicKey;

console.log("private key = " + privateKey);
console.log("public key = " + publicKey);

```

```
{% endtab %}

{% tab title="Go" %}

go
privateKey, err := hedera.GenerateEcdsaPrivateKey()
if err != nil {
    panic(err)
}

publicKey := privateKey.PublicKey()

fmt.Printf("private key = %v\n", privateKey)
fmt.Printf("public key = %v\n", publicKey)
```

```
{% endtab %}
{% endtabs %}
```

Sample Output:

```
private key =
3030020100300706052b8104000a04220420818c50766e025db403416421cb4a16d26ab0044b7f1a
1e45513cef2c86123b91
public key =
302d300706052b8104000a0322000224d3700dc68fc9061457c5f50b66442c73367f7d0b1d5a7e3a
1903e352ca217c
```

import-an-existing-key.md:

Import an existing key

Construct keys in another format to a key representation or import keys from a file.

Method	Type
PrivateKey.fromString(<code>&#x3C;privateKey></code>)	String
Constructs a private key string to PrivateKey	String
PublicKey.fromString(<code>&#x3C;publicKey></code>)	String
Constructs a public key string to PublicKey	String
PrivateKey.fromStringECDSA(<code>&#x3C;privateKey></code>)	String
Constructs an ECDSA key from a private key string	String
PublicKey.fromStringECDSA(<code>&#x3C;publicKey></code>)	String
Constructs an ECDSA public key from a public key string	String
PrivateKey.fromBytesECDSA(<code>&#x3C;privateKey></code>)	byte[]
Constructs an ECDSA key from a private key bytes	byte[]
PublicKey.fromBytesECDSA(<code>&#x3C;publicKey></code>)	byte[]
Constructs an ECDSA public key from a public key bytes	String
PrivateKey.fromStringED25519(<code>&#x3C;privateKey></code>)	String
Constructs an ED25519 key from a private key string	String
PublicKey.fromStringED25519(<code>&#x3C;publicKey></code>)	String
Constructs an ED25519 public key from a public key string	byte[]
PrivateKey.fromBytesED25519(<code>&#x3C;privateKey></code>)	byte[]
Constructs an ED25519 key from a private key bytes	byte[]
PublicKey.fromBytesED25519(<code>&#x3C;publicKey></code>)	byte[]
Constructs an ED25519 public key from a public key bytes	byte[]
PrivateKey.fromBytes(<code>&#x3C;privateKey></code>)	byte[]
Constructs a private key from bytes to	

<code>PrivateKey.fromBytes(&#x3C;publicKey>)/</code>	
<code>byte []</code>	Constructs a public key from bytes to
<code>PublicKey.fromPem(&#x3C;pemEncoded>)/</code>	
<code>String</code>	Parse a private key from a PEM encoded
<code>string</code>	
<code>PrivateKey.fromPem(&#x3C;encodedPem,</code>	
<code>password>)/</code>	<code>String, String</code>
	Parse a private key from a PEM
	encoded string. The private key may be encrypted, e.g. if it was generated by
	OpenSSL.
<code>PrivateKey.readPem(&#x3C;pemfile>)/</code>	
<code>Reader</code>	Parse a private key from a PEM encoded
<code>reader</code>	
<code>PrivateKey.readPem(&#x3C;pemFile,</code>	
<code>password>)/</code>	<code>Reader, String</code>
	Parse a private key from a PEM
	encoded stream. The key may be encrypted, e.g. if it was generated by
	OpenSSL.

```
{% tabs %}
{% tab title="Java" %}
java
//Converts a private key string to PrivateKey
PrivateKey privateKey =
PrivateKey.fromStringED25519("302e020100300506032b657004220420d763df96caaabf192c
67326e87c32a1ae4571f739022c77d2acaae5dd09cfb13");

//The public key associated with the private key
PublicKey publicKey =
PublicKey.fromStringED25519("302a300506032b65700321008f556741dcb5e144e5cabfce535
5ad5050ec7a6ea15787a5fd759d616e047d24");
```

```
{% endtab %}

{% tab title="JavaScript" %}
javascript
//Converts a private key string to PrivateKey
const privateKey =
PrivateKey.fromStringED25519("302e020100300506032b657004220420d763df96caaabf192c
67326e87c32a1ae4571f739022c77d2acaae5dd09cfb13");

//The public key associated with the private key
const publicKey =
PublicKey.fromStringED25519("302a300506032b65700321008f556741dcb5e144e5cabfce535
5ad5050ec7a6ea15787a5fd759d616e047d24");
```

```
{% endtab %}

{% tab title="Go" %}
go
//Converts a private key string to PrivateKey
privateKey, err :=
hedera.PrivateKeyFromStringEd25519("302e020100300506032b65700422042012a4a4add3d8
85bd61d7ce5cff88c5ef2d510651add00a7f64cb90de3359b105")
```

```
if err != nil {
    panic(err)
}

//The public key associated with the private key
publicKey, err :=
hedera.PublicKeyFromString("302a300506032b6570032100d292412f1c86507224c1db656050
c2162c91983540d608f6a31e9b43359bc5e")
```

```
if err != nil {
    panic(err)
}

{% endtab %}
```

```
{% endtabs %}
```

```
# README.md:
```

Keys

```
# recover-keys-from-a-mnemonic-phrase.md:
```

Recover keys from a mnemonic phrase

Recover private keys from a mnemonic phrase.

```
<table data-header-hidden><thead><tr><th></th><th>
width="145.33333333333331"></th><th></th></tr></thead><tbody><tr><td><strong>Met
hod</strong></td><td><strong>Type</strong></td><td><strong>Description</
strong></td></tr><tr><td><code>PrivateKey.fromMnemonic(&#x3C;mnemonic>)</
code></td><td>Mnemonic</td><td>Recover a private key from a mnemonic phrase
compatible with the iOS and Android
wallets</td></tr><tr><td><code>PrivateKey.fromMnemonic(&#x3C;mnemonic,
passphrase>)</code></td><td>Mnemonic. String</td><td>Recover a private key from
a generated mnemonic phrase and a passphrase</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
//Use the mnemonic to recover the private key
PrivateKey privateKey = PrivateKey.fromMnemonic(mnemonic);
PublicKey publicKey = privateKey.publicKey();
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
java
//Use a recovered mnemonic to recover the private key
const recoveredMnemonic = await Mnemonic.fromString(mnemonic.toString());
const privateKey = await recoveredMnemonic.toPrivateKey();
```

```
//v2.0.5
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
recoveredKey, err := hedera.PrivateKeyFromMnemonic(mnemonic, "")
publicKey := recoveredKey.PublicKey()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

```
# associate-tokens-to-an-account.md:
```

Associate tokens to an account

Associates the provided Hedera account with the provided Hedera token(s). Hedera accounts must be associated with a fungible or non-fungible token first before you can transfer tokens to that account. When you transfer a custom fungible or

non-fungible token to the alias account ID, the token association step is skipped and the account will automatically be associated with the token upon creation. In the case of NON\FUNGIBLE Type, once an account is associated, it can hold any number of NFTs (serial numbers) of that token type. The Hedera account that is associated with a token is required to sign the transaction.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALID\TOKEN\REF.

If any of the provided tokens has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an association between the provided account and any of the tokens already exists, the transaction will resolve to TOKEN\ALREADY\ASSOCIATED\TO\ACCOUNT.

If the provided account's associations count exceeds the constraint of maximum token associations per account, the transaction will resolve to TOKENS\PER\ACCOUNT\LIMIT\EXCEEDED.

On success, associations between the provided account and tokens are made and the account is ready to interact with the tokens.

{% hint style="info" %}

There is currently no limit on the number of token IDs that can be associated with an account (reference HIP-367). Still, you can see TOKENS\PER\ACCOUNT\LIMIT\EXCEEDED responses for pre-HIP-367 transactions.

{% endhint %}

Transaction Signing Requirements

The key of the account the token is being associated to
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	
-----	-----	
setAccountId(<accountId>)	AccountId	The account to be associated
with the provided tokens	Required	
setTokenIds(<tokens>)	List \<TokenId>	The tokens to be associated with
the provided account	Required	

{% tabs %}

{% tab title="Java" %}

java

//Associate a token to an account

```
TokenAssociateTransaction transaction = new TokenAssociateTransaction()  
    .setAccountId(accountId)  
    .setTokenIds(Collections.singletonList(tokenId));
```

//Freeze the unsigned transaction, sign with the private key of the account that is being associated to a token, submit the transaction to a Hedera network

```
TransactionResponse txResponse =  
transaction.freezeWith(client).sign(accountKey).execute(client);
```

//Request the receipt of the transaction

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status " +transactionStatus);
//v2.0.4

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Associate a token to an account and freeze the unsigned transaction for
signing
const transaction = await new TokenAssociateTransaction()
    .setAccountId(accountId)
    .setTokenIds([tokenId])
    .freezeWith(client);

//Sign with the private key of the account that is being associated to a token
const signTx = await transaction.sign(accountKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Associate the token to an account and freeze the unsigned transaction for
signing
transaction, err := hedera.NewTokenAssociateTransaction().
    SetAccountID(accountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the private key of the account that is being associated to a token,
submit the transaction to a Hedera network
txResponse, err = transaction.Sign(accountKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

```

```
//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}
```

atomic-swaps.md:

Atomic swaps

An atomic swap is when you swap tokens between two accounts without using a third-party intermediary, such as a centralized exchange or custody provider, to facilitate the transfer. Native tokens issued using the Hedera Token Service (HTS) can be swapped with another or with HBAR in a single transaction using the TransferTransaction API call. For each atomic swap within a single transaction, you'll need to designate an account to be debited (-) any number of tokens and the corresponding account which will receive those tokens.

Signing Requirements

The private keys for the accounts which are being debited tokens are required to sign the transaction.

```
{% hint style="info" %}
Hedera accounts must be associated with the specified token before you can
transfer a token to their account. Please see how to associate a token to an
account here.
{% endhint %}
```

```
{% tabs %}
{% tab title="Java" %}
java
//Atomic swap between a Hedera Token Service token and hbar
TransferTransaction atomicSwap = new TransferTransaction()
    .addHbarTransfer(accountId1, new Hbar(-10))
    .addHbarTransfer(accountId2, new Hbar(10))
    .addTokenTransfer(tokenId, accountId2, -1)
    .addTokenTransfer(tokenId, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
TransactionResponse txResponse =
atomicSwap.sign(accountKey1).sign(accountKey2).execute(client);
```

//-----OR-----

```
//Atomic swap between two hedera Token Service created tokens
TransferTransaction atomicSwap = new TransferTransaction()
    .addTokenTransfer(tokenId1, accountId1, -1)
    .addTokenTransfer(tokenId1, accountId2, 1)
    .addTokenTransfer(tokenId2, accountId2, -1)
    .addTokenTransfer(tokenId2, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
TransactionResponse txResponse =
atomicSwap.sign(accountKey1).sign(accountKey2).execute(client);
```

```

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Atomic swap between a Hedera Token Service token and hbar
const atomicSwap = await new TransferTransaction()
    .addHbarTransfer(accountId1, new Hbar(-10))
    .addHbarTransfer(accountId2, new Hbar(10))
    .addTokenTransfer(tokenId, accountId2, -1)
    .addTokenTransfer(tokenId, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
const txResponse = await (await (await
atomicSwap.sign(accountKey1)).sign(accountKey2)).execute(client);

//-----OR-----

//Atomic swap between two hedera Token Service created tokens
const atomicSwap = await new TransferTransaction()
    .addTokenTransfer(tokenId1, accountId1, -1)
    .addTokenTransfer(tokenId1, accountId2, 1)
    .addTokenTransfer(tokenId2, accountId2, -1)
    .addTokenTransfer(tokenId2, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
const txResponse = await (await (await
atomicSwap.sign(accountKey1)).sign(accountKey2)).execute(client);

{% endtab %}

{% tab title="Go" %}
go
//Atomic swap between a Hedera Token Service token and hbar
atomicSwap, err := hedera.NewTransferTransaction().
    AddHbarTransfer(accountId1, hedera.NewHbar(-10)).
    AddHbarTransfer(accountId2, hedera.NewHbar(10)).
    AddTokenTransfer(tokenId, accountId2, -1).
    AddTokenTransfer(tokenId, accountId1, 1).
    FreezeWith(client)

txResponse, err :=
atomicSwap.Sign(accountKey1).Sign(accountKey2).Execute(client)

//-----OR-----

//Atomic swap between two hedera Token Service created tokens
atomicSwap, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId1, accountId1, -1).
    AddTokenTransfer(tokenId1, accountId2, 1).
    AddTokenTransfer(tokenId2, accountId2, -1).
    AddTokenTransfer(tokenId2, accountId1, 1).
    FreezeWith(client)

txResponse, err :=
atomicSwap.Sign(accountKey1).Sign(accountKey2).Execute(client)

{% endtab %}
{% endtabs %}

```

burn-a-token.md:

Burn a token

Burns fungible and non-fungible tokens owned by the Treasury Account. If no Supply Key is defined, the transaction will resolve to TOKEN\HAS\NO\SUPPLY\KEY.

The operation decreases the Total Supply of the Token.

Total supply cannot go below zero.

The amount provided must be in the lowest denomination possible.

Example: Token A has 2 decimals. In order to burn 100 tokens, one must provide an amount of 10000. In order to burn 100.55 tokens, one must provide an amount of 10055.

This transaction accepts zero unit token burn operations for fungible tokens (HIP-564)

Transaction Signing Requirements

Supply key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	-----
-----	-----	-----
setTokenId(<tokenId>)	TokenId	The ID of the token to burn supply
Required		
setAmount(<amount>)	long	The number of tokens to burn (fungible tokens)
		Optional
setSerials(<serials>)	List<long>	Applicable to tokens of type NONFUNGIBLEUNIQUE. The list of NFT serial IDs to burn.
		Optional
addSerial(<serial>)	long	Applicable to tokens of type NONFUNGIBLEUNIQUE. The serial ID to burn.
		Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Burn 1,000 tokens
TokenBurnTransaction transaction = new TokenBurnTransaction()
    .setTokenId(tokenId)
    .setAmount(1000);

//Freeze the unsigned transaction, sign with the supply private key of the
token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(supplyKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
```

```

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Burn 1,000 tokens and freeze the unsigned transaction for manual signing
const transaction = await new TokenBurnTransaction()
    .setTokenId(tokenId)
    .setAmount(1000)
    .freezeWith(client);

//Sign with the supply private key of the token
const signTx = await transaction.sign(supplyKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Burn 1,000 tokens and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenBurnTransaction().
    SetTokenID(tokenId).
    SetAmount(1000).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the supply private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(supplyKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

```

```
{% endtab %}  
{% endtabs %}
```

custom-token-fees.md:

Custom token fees

When creating a token, you can configure up to 10 custom fees, automatically disbursed to specified fee collector accounts each time the token is transferred programmatically. These fees can be fixed, fractional, or royalty-based, offering revenue generation, profit-sharing, and behavior incentivization for creators. This guide is your comprehensive resource for understanding types, implementation, and best practices for custom fees on Hedera.

Types of Custom Fees

Fixed Fee: Paid by the sender of the fungible or non-fungible tokens. A fixed fee transfers a set amount to a fee collector account each time a token is transferred, independent of the transfer size. This fee can be collected in HBAR or another Hedera token but not in NFTs.

Fractional Fee: Take a specific portion of the transferred fungible tokens, with optional minimum and maximum limits. The token receiver (fee collector account) pays these fees by default. However, if `netoftransfers` is set to true, the sender pays the fees and the receiver collects the full token transfer amount. If this field is set to false, the receiver pays for the token custom fees and gets the remaining token balance.

Royalty Fee: Paid by the receiver account that is exchanging the fungible value for the NFT. When the NFT sender does not receive any fungible value, the fallback fee is charged to the NFT receiver.

```
{% hint style="info" %}
```

Note: In addition to the custom token fee payment, the sender account must pay for the token transfer transaction fee in HBAR. The "Payment of Custom Fees & Transaction Fees in HBAR" section below covers the distinction between custom fees and transaction fees.

```
{% endhint %}
```

Implementation Methods

Fixed Fee

A fixed fee entails transferring a specified token amount to predefined fee collector accounts each time a token transfer is initiated. This fee amount doesn't depend on the volume of tokens being transferred. The creator has the flexibility to collect the fee in HBAR or another fungible Hedera token. However, it's important to note that NFTs cannot be used as a token type to collect this fee. A custom fixed fee can be set for fungible and non-fungible token types.

```
<table><thead><tr><th  
width="409">Constructor</th><th>Description</th></tr></thead><tbody><tr><td><code>  
new CustomFixedFee()</code></td><td>Initializes the  
<code>CustomFixedFee</code> object</td></tr></tbody></table>
```

```
java  
new CustomFixedFee()
```

Methods	Description
Requirement	
<code>setFeeCollectorAccountId</code>	Sets the fee collector account ID that collects the fee.
AccountID	AccountID
Required	
<code>setHbarAmount</code>	Set the amount of HBAR to be collected.
HBAR	HBAR
Optional	
<code>setAmount</code>	Sets the amount of tokens to be collected as the fee.
int64	
Optional	
<code>setDenominatingTokenId</code>	The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.
TokenID	TokenID
Optional	
<code>setAllCollectorsAreExempt</code>	If true, exempts all the token's fee collector accounts from this fee.
boolean	
Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create a custom token fixed fee
new CustomFixedFee()
    .setAmount(1) // 1 token is transferred to the fee collecting account each
time this token is transferred
    .setDenominatingTokenId(tokenId) // The token to charge the fee in
    .setFeeCollectorAccountId(feeCollectorAccountId); // 1 token is sent to this
account everytime it is transferred

//Version: 2.0.143
```

```
{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a custom token fixed fee
new CustomFixedFee()
    .setAmount(1) // 1 token is transferred to the fee collecting account each
time this token is transferred
    .setDenominatingTokenId(tokenId) // The token to charge the fee in
    .setFeeCollectorAccountId(feeCollectorAccountId); // 1 token is sent to this
account everytime it is transferred

//Version: 2.0.30
```

```
{% endtab %}

{% tab title="Go" %}
go
//Create a custom token fixed fee
[]hedera.Fee{
    hedera.NewCustomFixedFee().
        SetAmount(1). // 1 token is transferred to the fee collecting
account each time this token is transferred
        SetDenominatingTokenID(tokenId). // The token to charge the fee in
        SetFeeCollectorAccountID(feeCollectorAccountId) // 1 token is sent
to this account everytime it is transferred
},
}
```


//Version: 2.1.16

{% endtab %}
{% endtabs %}

Fractional Fee

Fractional fees involve the transfer of a specified fraction of the tokens' total value to the designated fee collector account. You can set a custom fractional fee and impose minimum and maximum fee limits per transfer transaction. The fractional fee has to be less than or equal to 1. It cannot exceed the fractional range of a 64-bit signed integer. Applicable to fungible tokens only.

Methods	Description	Type	Requirement
<code>setFeeCollectorAccountId</code>	Sets the fee collector account ID that collects the fee.	AccountId	Required
<code>setNumerator</code>	Sets the numerator of the fraction.	long	Required
<code>setDenominator</code>	Sets the denominator of the fraction. Cannot be zero.	long	Required
<code>setMax</code>	The maximum fee that can be charged, regardless of the fractional value.	long	Optional
<code>setMin</code>	The minimum fee that can be charged, regardless of the fractional value.	long	Optional
<code>setAssessmentMethod</code>	If true, sender pays fees and the receiver collects the full token transfer amount. If false, receiver pays fees and gets remaining token balance.	boolean	Optional
<code>FeeAssessmentMethod</code>			
<code>setAllCollectorsAreExempt</code>	If true, exempts all the token's fee collector accounts from this fee.	boolean	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create a custom token fractional fee
new CustomFractionalFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFeeCollectorAccountId(feeCollectorAccountId); // The account collecting
the 10% custom fee each time the token is transferred

//Version: 2.0.14
```

{% endtab %}

{% tab title="JavaScript" %}

```
javascript
//Create a custom token fractional fee
new CustomFractionalFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFeeCollectorAccountId(feeCollectorAccountId); // The account collecting
```

the 10% custom fee each time the token is transferred

```
//Version: 2.0.30
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
//Create a custom token fractional fee
```

```
[]hedera.Fee{
    hedera.NewCustomFractionalFee().
        SetNumerator(1). // The numerator of the fraction
        SetDenominator(10). // The denominator of the fraction
        SetFeeCollectorAccountID(feeCollectorAccountId), // The account
collecting the 10% custom fee each time the token is transferred
}
```

```
//Version: 2.1.16
```


```
{% endtab %}
```

```
{% endtabs %}
```

Royalty Fee

The royalty fee is assessed and applied each time the ownership of an NFT is transferred and is a fraction of the value exchanged for the NFT. If no value is exchanged for the NFT, a fallback fee can be imposed on the receiving account. This fee type only applies to non-fungible tokens.

```
{% hint style="info" %}
```

 NOTE: Royalty fees are strictly a convenience feature. The network can't enforce royalties if counterparties decide to split their NFT exchange into separate transactions. The NFT sender and receiver must both sign a single CryptoTransfer to ensure the proper application of royalties. There is an open HIP discussion about broadening the class of transactions for which the network automatically collects royalties. If this topic interests or concerns you, your participation in the discussion is welcome.

```
{% endhint %}
```

```
<table><thead><tr><th
width="394">Constructor</th><th>Description</th></tr></thead><tbody><tr><td><code>new CustomRoyaltyFee()</code></td><td>Initializes the
<code>CustomRoyaltyFee</code> object</td></tr></tbody></table>
```

```
java
```

```
new CustomRoyaltyFee()
```

```
<table><thead><tr><th width="287.33333333333326">Methods</th><th
width="223">Description</th><th width="110" align="center">Type</th><th
align="center">Requirement</th></tr></thead><tbody><tr><td><code>setFeeCollector
AccountId</code></td><td>Sets the fee collector account ID that collects the
fee.</td><td align="center"><a href=".../deprecated/sdks/specialized-
types.md#accountid">AccountId</a></td><td
align="center">Required</td></tr><tr><td><code>setNumerator</code></td><td>Sets
the numerator of the fraction.</td><td align="center">long</td><td
align="center">Required</td></tr><tr><td><code>setDenominator</code></td><td>Sets the denominator of the fraction.</td><td align="center">long</td><td
align="center">Required</td></tr><tr><td><code>setFallbackFee</code></td><td>If
present, the fixed fee to assess to the NFT receiver when no fungible value is
exchanged with the sender</td><td align="center"><a
href=".../hedera-api/token-service/customfees/fixedfee.md">FixedFee</a></td><td
align="center">Optional</td></tr><tr><td><code>setAllCollectorsAreExempt</code></td><td></td><td align="center"></td><td align="center"></td></tr></tbody></table>
```

code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr></tbody></table>

```
{% tabs %}
{% tab title="Java" %}
java
//Create a royalty fee
new CustomRoyaltyFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1)) // The
fallback fee
    .setFeeCollectorAccountId(feeCollectorAccountId))) // The account that will
receive the royalty fee

// v2.0.14

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a royalty fee
new CustomRoyaltyFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1)) // The
fallback fee
    .setFeeCollectorAccountId(feeCollectorAccountId))) // The account that will
receive the royalty fee

// v2.0.30

{% endtab %}

{% tab title="Go" %}
go
//Create a royalty fee
[]hedera.Fee{
    hedera.NewCustomRoyaltyFee().
        SetFeeCollectorAccountID(feeCollectorAccountId). // The account that
will receive the royalty fee
        SetNumerator(1). // The numerator of the fraction
        SetDenominator(10). // The denominator of the fraction
        SetFallbackFee( // The fallback fee
            hedera.NewCustomFixedFee().
                SetFeeCollectorAccountID(feeCollectorAccountId).
                SetAmount(1),
        ),
}

// v2.1.16

{% endtab %}
{% endtabs %}
```

Payment of Custom Fees vs. Transaction Fees in HBAR

Understanding the difference between custom fees and standard transaction fees in HBAR is crucial for token issuers and developers working with Hedera. Custom fees are designed to enforce complex fee structures, such as royalties and fractional ownership. These fees can be fixed, fractional, or royalty-based and

are usually paid in the token being transferred, although other Hedera tokens or HBAR can also be used. You can configure up to 10 custom fees automatically disbursed to designated fee collector accounts.

On the other hand, transaction fees in HBAR serve a different purpose: they compensate network nodes for processing transactions. These fees are uniform across all transaction types and are paid exclusively in HBAR. Unlike custom fees, which can be configured by the user, transaction fees are fixed by the network.

The key differences lie in their flexibility, payee, currency, and configurability. Custom fees offer greater flexibility and can be paid to any account in various tokens, and are user-defined. Transaction fees are network-defined, less flexible, and go solely to network nodes, paid only in HBAR.

Fee Exemptions

Fee collector accounts can be exempt from paying custom fees. To enable this, you need to set the exemption during the creation of the custom fees (HIP-573). If not enabled, custom fees will only be exempt for an account if that account is set as a fee collector.

Limits and Constraints

When it comes to setting custom fees, there are a few limits and constraints to keep in mind:

- First, fees cannot be set to a negative value.
- Each token can have up to 10 different custom fees.
- Additionally, the treasury account for a given token is automatically exempt from paying these custom transaction fees.
- The system also permits, at most, two "levels" of custom fees. That means a token being transferred might require fees in another token that also has its own fee schedule; however, this can only be nested two layers deep to prevent excessive complexity.

define-a-token.md:

Create a token

```
{% hint style="info" %}
Check out "Getting Started with the Hedera Token Service" video tutorial in
JavaScript here.
{% endhint %}
```

Create a new fungible or non-fungible token (NFT) on the Hedera network. After you submit the transaction to the Hedera network, you can obtain the new token ID by requesting the receipt.

You can also create, access, or transfer HTS tokens using smart contracts - see Hedera Service Solidity Libraries and Supported ERC Token Standards.

```
{% hint style="warning" %}
Token Keys
```

If any of the token key types (KYC key, Wipe key, etc) are not set during the creation of the token, you will not be able to update the token and add them in the future

If any of the token key types (KYC key, Wipe key, etc) are set during the creation of the token, you will not be able to remove them in the future

```
{% endhint %}
```

NFTs

For non-fungible tokens, the token ID represents a NFT class. Once the token is created, you will have to mint each NFT using the token mint operation.

{% hint style="warning" %}

Note: It is required to set the initial supply for an NFT to 0.

{% endhint %}

Token Properties

Property	Description
-----	-----
-----	-----
-----	-----
-----	-----
-----	-----
-----	-----
Name	Set the publicly visible name of the token. The token name is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (NUL). The token name is not unique. Maximum of 100 characters.
Token Type	The type of token to create. Either fungible or non-fungible.
Symbol	The publicly visible token symbol. Set the publicly visible name of the token. The token symbol is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (NUL). The token symbol is not unique. Maximum of 100 characters.
Decimal	The number of decimal places a token is divisible by. This field can never be changed.
Initial Supply	Specifies the initial supply of fungible tokens to be put in circulation. The initial supply is sent to the Treasury Account. The maximum supply of tokens is 9,223,372,036,854,775,807($2^{63}-1$) tokens and is in the lowest denomination possible. For creating an NFT, you must set the initial supply to 0.
Treasury Account	The account which will act as a treasury for the token. This account will receive the specified initial supply and any additional tokens that are minted. If tokens are burned, the supply will decreased from the treasury account.
Admin Key	The key which can perform token update and token delete operations on the token. The admin key has the authority to change the supply key, freeze key, pause key, wipe key, and KYC key. It can also update the treasury account of the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted).
KYC Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
Freeze Key	The key which can sign to freeze or unfreeze an account for token transactions. If empty, freezing is not possible.
Wipe Key	The key which can wipe the token balance of an account. If empty, wipe is not possible.
Supply Key	The key which can change the total supply of a token.

This key is used to authorize token mint and burn transactions. If this is left empty, minting/burning tokens is not possible.

| Fee Schedule Key | The key which can change the token's custom fee schedule. It must sign a TokenFeeScheduleUpdate transaction. A custom fee schedule token without a fee schedule key is immutable.

| Pause Key | The key which has the authority to pause or unpause a token. Pausing a token prevents the token from participating in all transactions.

| Custom Fees | Custom fees to charge during a token transfer transaction that transfers units of this token. Custom fees can either be fixed, fractional, or royalty fees. You can set up to a maximum of 10 custom fees.

| Max Supply | <p>For tokens of type `FUNGIBLECOMMON` - the maximum number of tokens that can be in circulation.
For tokens of type `NONFUNGIBLEUNIQUE` - the maximum number of NFTs (serial numbers) that can be minted. This field can never be changed.
You must set the token supply type to FINITE if you set this field.</p>

| Supply Type | Specifies the token supply type. Defaults to INFINITE.

| Freeze Default | The default Freeze status (frozen or unfrozen) of Hedera accounts relative to this token. If true, an account must be unfrozen before it can receive the token.

| Expiration Time | The epoch second at which the token should expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod. The default expiration time is 7,890,000 seconds (90 days).

| Auto Renew Account | An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval. This key is required to sign the transaction if present. Currently, rent is not enforced for tokens so auto-renew payments will not be made.

| Auto Renew Period | <p>The interval at which the auto-renew account will be charged to extend the token's expiry. The default auto-renew period is 7,890,000 seconds. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: AUTORENEWDURATIONNOTINRANGE.</p> |

| Memo | A short publicly visible memo about the token.

Transaction Signing Requirements

Treasury key is required to sign
Admin key, if specified
Transaction fee payer key

Transaction Fees

For fungible tokens, a CryptoTransfer fee is added to transfer the newly created token to the treasury account
Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type
Requirement	

-----		-----
setTokenName(<name>)	String	
Required		
setTokenType(<tokenType>)	TokenType	
Optional		
setTokenSymbol(<symbol>)	String	
Required		
setDecimals(<decimal>)	int	
Optional		
setInitialSupply(<initialSupply>)	int	
Optional		
setTreasuryAccountId(<treasury>)	AccountId	Required
setAdminKey(<key>)	Key	
Optional		
setKycKey(<key>)	Key	
Optional		
setFreezeKey(<key>)	Key	
Optional		
setWipeKey(<key>)	Key	
Optional		
setSupplyKey(<key>)	Key	
Optional		
setPauseKey(<key>)	Key	
Optional		
setFreezeDefault(<freeze>)	boolean	
Optional		
setExpirationTime(<expirationTime>)	Instant	
Optional		
setFeeScheduleKey(<key>)	Key	
Optional		
setCustomFees(<customFees>)	List<CustomFee>	Optional
setSupplyType(<supplyType>)	TokenSupplyType	
Optional		
setMaxSupply(<maxSupply>)	long	
Optional		
setTokenMemo(<memo>)	String	
Optional		
setAutoRenewAccountId(<account>)	AccountId	Optional
setAutoRenewPeriod(<period>)	Duration	
Optional		

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TokenCreateTransaction transaction = new TokenCreateTransaction()
    .setTokenName("Your Token Name")
    .setTokenSymbol("F")
    .setTreasuryAccountId(treasuryAccountId)
    .setInitialSupply(5000)
    .setAdminKey(adminKey.getPublicKey())
    .setMaxTransactionFee(new Hbar(30)); //Change the default max
transaction fee

//Build the unsigned transaction, sign with admin private key of the token, sign
with the token treasury private key, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).sign(treasuryKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the token ID from the receipt
```

```

tokenId tokenId = receipt.tokenId;

System.out.println("The new token ID is " + tokenId);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze for manual signing
const transaction = await new TokenCreateTransaction()
    .setTokenName("Your Token Name")
    .setTokenSymbol("F")
    .setTreasuryAccountId(treasuryAccountId)
    .setInitialSupply(5000)
    .setAdminKey(adminPublicKey)
    .setMaxTransactionFee(new Hbar(30)) //Change the default max transaction
fee
    .freezeWith(client);

//Sign the transaction with the token adminKey and the token treasury account
private key
const signTx = await (await transaction.sign(adminKey)).sign(treasuryKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the token ID from the receipt
const tokenId = receipt.tokenId;

console.log("The new token ID is " + tokenId);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction
tokenCreateTransaction, err := hedera.NewTokenCreateTransaction().
    SetTokenName("Your Token Name").
    SetTokenSymbol("F").
    SetTreasuryAccountID(treasuryAccountId).
    SetInitialSupply(1000).
    SetAdminKey(adminKey).
    SetMaxTransactionFee(hedera.NewHbar(30)). //Change the default max
transaction fee
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin private key of the token, sign with the token treasury
private key, sign with the client operator private key and submit the
transaction to a Hedera network
txResponse, err :=
tokenCreateTransaction.Sign(adminKey).Sign(treasuryKey).Execute(client)

```



```

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the token ID from the receipt
tokenId := receipt.TokenID

fmt.Printf("The new token ID is %v\n", tokenId)

//v2.1.0

{% endtab %}
{% endtabs %}

```

delete-a-token.md:

Delete a token

Deleting a token marks a token as deleted, though it will remain in the ledger. The operation must be signed by the specified Admin Key of the Token. If the Admin Key is not set, the Transaction will result in TOKEN\IS\IMMUTABLE. Once deleted update, mint, burn, wipe, freeze, unfreeze, grant KYC, revoke KYC and token transfer transactions will resolve to TOKEN\WAS\DELETED.

NFTs

You cannot delete a specific NFT. You can delete the class of the NFT specified by the token ID after you have burned all associated NFTs associated with the token class

Transaction Signing Requirements

Admin key
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	-----
-----	-----	-----
setTokenId(<tokenId>)	TokenId	The ID of the token to delete Required

```

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TokenDeleteTransaction transaction = new TokenDeleteTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the admin private key of the

```

```

account, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze the unsigned transaction for manual signing
const transaction = await new TokenDeleteTransaction()
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the admin private key of the token
const signTx = await transaction.sign(adminKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenDeleteTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin private key of the account, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(adminKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {

```

```

        panic(err)
    }

    //Get the transaction consensus status
    status := receipt.Status

    fmt.Printf("The transaction consensus status is %v\n", status)

    //v2.1.0

    {% endtab %}
    {% endtabs %}

```

disable-kyc-account-flag.md:

Disable KYC account flag

Revokes the KYC flag to the Hedera account for the given Hedera token. This transaction must be signed by the token's KYC Key. If this key is not set, you can submit a TokenUpdateTransaction to provide the token with this key.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKEN\HAS\NO\KYC\KEY. Once executed the Account is marked as KYC Revoked

Transaction Signing Requirements

KYC key
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	

setTokenId(<tokenId>)	TokenId	The token ID that is associated with the account to remove the KYC flag for
	Required	
setAccountId(<setAccountId>)	AccountId	The account ID that is associated with the account to remove the KYC flag
	Required	

```

{% tabs %}
{% tab title="Java" %}
java
//Remove the KYC flag from an account
TokenRevokeKycTransaction transaction = new TokenRevokeKycTransaction()
    .setTokenId(tokenId)
    .setAccountId(accountId);

```

```

//Freeze the unsigned transaction, sign with the kyc private key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(kycKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//Version: 2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Remove the KYC flag on account and freeze the transaction for signing
const transaction = await new TokenRevokeKycTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the kyc private key of the token
const signTx = await transaction.sign(kycKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Remove the KYC flag from an account and freeze the transaction for signing
transaction, err = hedera.NewTokenRevokeKycTransaction().
    SetTokenID(tokenId).
    SetAccountID(accountId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the kyc private key of the token, submit the transaction to a Hedera
network
txResponse, err := transaction.Sign(kycKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction

```

```

receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

dissociate-tokens-from-an-account.md:

Dissociate tokens from an account

Dissociates the provided Hedera account from the provided Hedera tokens. This transaction must be signed by the provided account's key. Once the association is removed, no token-related operation can be performed to that account. AccountBalanceQuery and AccountInfoQuery will not return anything related to the dissociated token.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALIDTOKENREF.

If an association between the provided account and any of the tokens does not exist, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If the provided account has a nonzero balance with any of the provided tokens, the transaction will resolve to TRANSACTIONREQUIRESZEROTOKENBALANCES.

On success, associations between the provided account and tokens are removed.

{% hint style="info" %}

The account is required to have a zero balance of the token you wish to dissociate. If a token balance is present, you will receive a TRANSACTIONREQUIRESZEROTOKENBALANCES error.

{% endhint %}

Transaction Signing Requirements

The key of the account the token is being dissociated with
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	

setTokenIds(<tokenId>)	TokenId	The tokens to be dissociated with the
provided account Required		
setAccountId(<accountId>)	AccountId	The account to be dissociated with the

provided tokens | Required |

```
{% tabs %}
{% tab title="Java" %}
java
//Dissociate a token from an account
TokenDissociateTransaction transaction = new TokenDissociateTransaction()
    .setAccountId(accountId)
    .setTokenIds(tokenId);

//Freeze the unsigned transaction, sign with the private key of the account that
is being dissociated from a token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(accountKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Dissociate a token from an account and freeze the unsigned transaction for
signing
const transaction = await new TokenDissociateTransaction()
    .setAccountId(accountId)
    .setTokenIds([tokenId])
    .freezeWith(client);

//Sign with the private key of the account that is being associated to a token
const signTx = await transaction.sign(accountKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Dissociate the token from an account and freeze the unsigned transaction for
signing
transaction, err := hedera.NewTokenDissociateTransaction().
    SetAccountID(accountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}
```

```

}

//Sign with the private key of the account that is being associated to a token,
submit the transaction to a Hedera network
txResponse, err = transaction.Sign(accountKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

enable-kyc-account-flag.md:

Enable KYC account flag

Grants KYC to the Hedera accounts for the given Hedera token. This transaction must be signed by the token's KYC Key.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKEN\HAS\NO\KYC\KEY. Once executed the Account is marked as KYC Granted.

Transaction Signing Requirements

KYC key
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Required		
-----	-----	
-----	-----	

```

| setTokenId(<tokenId>)      | TokenId    | The token for this account to have
passed KYC | Required |
| setAccountId(<accountId>) | AccountId  | The account for this token to have
passed KYC | Required |

{% tabs %}
{% tab title="Java" %}
java
//Enable KYC flag on account
TokenGrantKycTransaction transaction = new TokenGrantKycTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the kyc private key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(kycKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Enable KYC flag on account and freeze the transaction for manual signing
const transaction = await new TokenGrantKycTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the kyc private key of the token
const signTx = await transaction.sign(kycKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Enable KYC flag on account and freeze the transaction for manual signing
transaction, err = hedera.NewTokenGrantKycTransaction().
    SetAccountID(accountId).
    SetTokenID(tokenId).
    FreezeWith(client)

```



```

if err != nil {
    panic(err)
}

//Sign with the kyc private key of the token, submit the transaction to a Hedera
network
txResponse, err := transaction.Sign(kycKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

errors.md:

Network Response Messages

Network response messages and their descriptions.

Network Response	Description
-----	-----
ACCOUNTFROZENFORTOKEN and cannot transact with the token	The account is frozen
TOKENSPERACCOUNTLIMITEXCEEDED token relations for a given account is exceeded	The maximum number of
INVALIDTOKENID or does not exist	The token is invalid
INVALIDTOKENDECIMALS decimals	Invalid token
INVALIDTOKENINITIALSUPPLY supply	Invalid token initial
INVALIDTREASURYACCOUNTFORTOKEN not exist or is deleted	Treasury account does
INVALIDTOKENSYMBOL UTF-8 capitalized alphabetical string	Token Symbol is not
TOKENHASNOFREEZEKEY on a token	Freeze key is not set

TRANSFERSNOTZEROSUMFORTOKEN list are not net-zero	Amounts in the transfer
MISSINGTOKENSYMBOL provided	Token Symbol is not
TOKENSYMBOLTOOLONG long	Token Symbol is too
ACCOUNTKYCNOTGRANTEDFORTOKEN the account does not have KYC granted	KYC must be granted and
TOKENHASNOKYCKEY a token	KYC key is not set on
INSUFFICIENTTOKENBALANCE sufficient for the transaction	Token balance is not
TOKENWASDELETED cannot be executed on deleted token	Token transactions
TOKENHASNOSUPPLYKEY set on a token	The supply key is not
TOKENHASNOWIPEKEY set on a token	The wipe key is not
INVALIDTOKENMINTAMOUNT	Invalid mint amount
INVALIDTOKENBURNAMOUNT	Invalid burn amount
TOKENNOTASSOCIATEDTOACCOUNT associated with an account	Account has not been
CANNOTWIPETOKENTREASURYACCOUNT operation on treasury account	Cannot execute wipe
INVALIDKYCKEY	Invalid kyc key
INVALIDWIPEKEY	Invalid wipe key
INVALIDFREEZEKEY	Invalid freeze key
INVALIDSUPPLYKEY	Invalid supply key
MISSINGTOKENNAME provided	Token Name is not
TOKENNAMETOOLONG long	Token Name is too
INVALIDWIPINGAMOUNT amount must not be negative, zero or bigger than the token holder balance	The provided wipe
TOKENISIMMUTABLE have Admin key set, thus update/delete transactions cannot be performed	The token does not
TOKENALREADYASSOCIATEDTOACCOUNT operation specified a token already associated with the account	An associateToken
TRANSACTIONREQUIRESZEROTOKENBALANCES is invalid until all token balances for the target account are zero	An attempted operation
ACCOUNTISTREASURY operation is invalid because the account is a treasury	An attempted

	TOKENIDREPEATEDINTOKENLIST	Same TokenIDs present
in the token list		
	TOKENTRANSFERLISTSIZELIMITEXCEEDED	Exceeded the number of
token transfers (both from and to) allowed for token transfer list		
	EMPTYTOKENTRANSFERBODY	
TokenTransfersTransactionBody has no TokenTransferList		
	EMPTYTOKENTRANSFERACCOUNTAMOUNTS	
TokenTransfersTransactionBody has a TokenTransferList with no AccountAmounts		
	FRACTIONDIVIDESBYZERO	A custom fractional
fee set a denominator of zero		
	INSUFFICIENTPAYERBALANCEFORCUSTOMFEE	The transaction payer
could not afford a custom fee		
	CUSTOMFEESLISTTOOLONG	The customFees list is
longer than allowed limit 10		
	INVALIDCUSTOMFEECOLLECTOR	Any of the
feeCollector accounts for customFees is invalid		
	INVALIDTOKENIDINCUSTOMFEES	Any of the token Ids in
customFees is invalid		
	TOKENNOTASSOCIATEDTOFEECOLLECTOR	Any of the token Ids in
customFees are not associated to feeCollector		
	CUSTOMFEENOTFULLYSPECIFIED	A custom fee schedule
entry did not specify either a fixed or fractional fee		
	CUSTOMFEEMUSTBEPOSITIVE	Only positive fees may
be assessed at this time		
	TOKENHASNOFEESCHEDULEKEY	Fee schedule key is not
set on token		
	CUSTOMFEEOUTSIDENUMERICRANGE	A fractional custom
fee exceeded the range of a 64-bit signed integer		
	INVALIDCUSTOMFRACTIONALFEESUM	The sum of all custom
fractional fees must be strictly less than 1		
	FRACTIONALFEEMAXAMOUNTLESSTHANMINAMOUNT	Each fractional custom
fee must have its maximum\amount, if specified, at least its minimum\amount		
	CUSTOMSCHEDULEALREADYHASNOFEES	A fee schedule update
tried to clear the custom fees from a token whose fee schedule was already empty		
	CUSTOMFEEDENOMINATIONMUSTBEFUNGIBLECOMMON	Only tokens of type
FUNGIBLE\COMMON can be used as fee schedule denominations		
	CUSTOMFRACTIONALFEEONLYALLOWEDFORFUNGIBLECOMMON	Only tokens of type
FUNGIBLE\COMMON can have fractional fees		
	INVALIDCUSTOMFEESCHEDULEKEY	The provided custom
fee schedule key was invalid		
	ACCOUNTAMOUNTTRANSFERONLYALLOWEDFORFUNGIBLECOMMON	An AccountAmount token
transfers list referenced a token type other than FUNGIBLE\COMMON		
	INVALIDTOKENMINTMETADATA	The requested token
mint metadata was invalid		
	INVALIDTOKENBURNMETADATA	The requested token
burn metadata was invalid		

PAYERACCOUNTDELETED	The payer account
has been marked as deleted	
CUSTOMFEECHARGINGEXCEEDEDMAXRECURSIONDEPTH	The reference chain of
custom fees for a transferred token exceeded the maximum length of 2	
CUSTOMFEECHARGINGEXCEEDEDMAXACCOUNTAMOUNTS	More than 20 balance
adjustments were to satisfy a CryptoTransfer and its implied custom fee payments	
INSUFFICIENTSENDERACCOUNTBALANCEFORCUSTOMFEE	The sender account in
the token transfer transaction could not afford a custom fee	
SERIALNUMBERLIMITREACHED	Currently no more
than 4,294,967,295 NFTs may be minted for a given unique token type	
CUSTOMROYALTYFEEONLYALLOWEDFORNONFUNGIBLEUNIQUE	Only tokens of type NON\
FUNGIBLE\UNIQUE can have royalty fees	
TOKENISPAUSED	Token is paused.
This Token cannot be a part of any kind of Transaction until unpaused.	
TOKENHASNOPAUSEKEY	Pause key is not set
on token	
INVALIDPAUSEKEY	The provided pause
key was invalid	

freeze-an-account.md:

Freeze an account

Freezes transfers of the specified token for the account. The transaction must be signed by the token's Freeze Key.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID. If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Frozen and will not be able to receive or send tokens unless unfrozen.

The operation is idempotent

Transaction Signing Requirements

Freeze key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
--------	------	-------------

Requirement		
setTokenId(<tokenId>)	TokenId	The token for this account to be frozen
Required		
setAccountId(<accountId>)	AccountId	The account to be frozen
Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Freeze an account from transferring a token
TokenFreezeTransaction transaction = new TokenFreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the sender freeze private key of
the token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(freezeKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.print("The transaction consensus status is " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Freeze an account from transferring a token
const transaction = await new TokenFreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the freeze key of the token
const signTx = await transaction.sign(freezeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Freeze an account from transferring a token
transaction, err = hedera.NewTokenFreezeTransaction().
    SetAccountID(accountId).
    SetTokenID(tokenId).
```

```

        FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the freeze private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(freezeKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

get-account-token-balance.md:

Get account token balance

To get the balance of tokens for an account, you can submit an account balance query. The account balance query will return the tokens the account holds in a list format.

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Method	Type	Requirement
setAccountId(<accountId>)	AccountId	Required

```

{% tabs %}
{% tab title="Java" %}
java
//Create the query
AccountBalanceQuery query = new AccountBalanceQuery()
    .setAccountId(accountId);

```

```

//Sign with the operator private key and submit to a Hedera network
AccountBalance tokenBalance = query.execute(client);

```

```

System.out.println("The token balance(s) for this account: "
+tokenBalance.tokens);

```

//v2.0.9

```

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Sign with the client operator private key and submit to a Hedera network
const tokenBalance = await query.execute(client);

console.log("The token balance(s) for this account: "
+tokenBalance.tokens.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the query
query := hedera.NewAccountBalanceQuery().
    SetAccountID(accountId)

//Sign with the client operator private key and submit to a Hedera network
tokenBalance, err := query.Execute(client)

if err != nil {
    panic(err)
}

fmt.Printf("The token balance(s) for this account: %v\n", tokenBalance)

//v2.1.0

{% endtab %}
{% endtabs %}

```

get-nft-token-info.md:

Get NFT info

A query that returns information about a non-fungible token (NFT). You request the info for an NFT by specifying the NFT ID.

Token Allowances

Only when a spender is set on an explicit NFT ID of a token, we return the spender ID in theTokenNftInfoQuery for the respective NFT. If approveTokenNftAllowanceAllSerials is used to approve all NFTs for a given token class and no NFT ID is specified, we will not return a spender ID for all the serial numbers of that token.

Query Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

```

{% hint style="warning" %}
Requesting NFT info by Token ID or Account ID is deprecated.
{% endhint %}

```

The request returns the following information:

Item	Description
NFT ID	The ID of the non-fungible token in x.y.z format.
Account ID	The account ID of the current owner of the NFT
Creation Time	The effective consensus timestamp at which the NFT was minted
Metadata	Represents the unique metadata of the NFT
Ledger ID	The ID of the network (mainnet, testnet, previewnet).
Spender ID	The spender account ID for the NFT. This is only returned if the NFT ID was specifically approved.

Methods

Method	Type	Description
setNftId(<nftId>)	NftId	Applicable only to tokens of type NONFUNGIBLEUNIQUE. Gets info on a NFT for a given TokenID (of type NONFUNGIBLEUNIQUE) and serial number.

```
{% tabs %}
{% tab title="Java" %}
java
//Returns the info for the specified NFT ID
List<TokenNftInfo> nftInfos = new TokenNftInfoQuery()
    .setNftId(nftId)
    .execute(client);
```

//v2.0.14

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Returns the info for the specified NFT ID
const nftInfos = await new TokenNftInfoQuery()
    .setNftId(nftId)
    .execute(client);
```

//v2.0.28

```
{% endtab %}
```

```
{% tab title="Go" %}
go
//Returns the info for the specified NFT ID
nftInfo, err := NewTokenNftInfoQuery().
    SetNftID(nftID).
    Execute(client)
```

//v2.1.16

```
{% endtab %}
```



```
{% endtabs %}
```

```
# get-token-info.md:
```

Get token info

Gets information about a fungible or non-fungible token instance. The token info query returns the following information:

Query Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Item	Description
TokenId	ID of the token instance
Token Type	The type of token (fungible or non-fungible)
Name characters	The name of the token. It is a string of ASCII only characters
Symbol alphabetical string	The symbol of the token. It is a UTF-8 capitalized alphabetical string
Decimals	The number of decimal places a token is divisible by
Total Supply circulation	The total supply of tokens that are currently in circulation
Treasury	The ID of the account which is set as Treasury
Custom Fees	The custom fee schedule of the token, if any
Fee Schedule Key	Fee schedule key, if any
Admin Key	The key which can perform update/delete operations on the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted)
KYC Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
Freeze Key	The key which can freeze or unfreeze an account for token transactions. If empty, freezing is not possible
Wipe Key	The key which can wipe token balance of an account. If empty, wipe is not possible
Supply Key	The key which can change the supply of a token. The key is used to sign Token Mint/Burn operations
Pause Key	The key that can pause or unpause the token from participating in transactions.

Pause Status	<p>Whether or not the token is paused.</p><p>>false = not paused</p><p>>true = paused</p>
Max Supply	The max supply of the token
Supply Type	The supply type of the token
Default Freeze Status	<p>The default Freeze status (not applicable = null, frozen = false, or unfrozen = true) of Hedera accounts relative to this token. FreezeNotApplicable is returned if Token Freeze Key is empty. Frozen is returned if Token Freeze Key is set and defaultFreeze is set to true. Unfrozen is returned if Token Freeze Key is set and defaultFreeze is set to false.</p><p>FreezeNotApplicable = null;</p><p>Frozen = true;</p><p>Unfrozen = false;</p>
Default KYC Status	<p>The default KYC status (KycNotApplicable or Revoked) of Hedera accounts relative to this token. KycNotApplicable is returned if KYC key is not set, otherwise Revoked.</p><p>KycNotApplicable = null;</p><p>Granted = false;</p><p>Revoked = true;</p>
Auto Renew Account	An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval
Auto Renew Period	The interval at which the auto-renew account will be charged to extend the token's expiry
Expiry	The epoch second at which the token will expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod
Ledger ID	The ID of the network the response came from. See HIP-198.
Memo	Short publicly visible memo about the token, if any

Methods

Method	Type	Requirement
setTokenId(<tokenId>)	TokenId	Required
<TokenInfo>.tokenId	TokenId	Optional
<TokenInfo>.name	String	Optional
<TokenInfo>.symbol	String	Optional
<TokenInfo>.decimals	int	Optional
<TokenInfo>.customFees	List<CustomFee>	Optional
<TokenInfo>.totalSupply	long	Optional
<TokenInfo>.treasuryAccountId	AccountId	Optional
<TokenInfo>.adminKey	Key	Optional
<TokenInfo>.kycKey	Key	Optional
<TokenInfo>.freezeKey	Key	Optional
<TokenInfo>.feeScheduleKey	Key	Optional
<TokenInfo>.wipeKey	Key	Optional
<TokenInfo>.supplyKey	Key	Optional
<TokenInfo>.defaultFreezeStatus	boolean	Optional
<TokenInfo>.defaultKycStatus	boolean	Optional
<TokenInfo>.isDeleted	boolean	Optional
<TokenInfo>.tokenType	TokenType	Optional
<TokenInfo>.supplyType	TokenSupplyType	Optional
<TokenInfo>.maxSupply	long	Optional
<TokenInfo>.pauseKey	Key	Optional
<TokenInfo>.pauseStatus	boolean	Optional
<TokenInfo>.autoRenewAccount	AccountId	Optional
<TokenInfo>.autoRenewPeriod	Duration	Optional

<TokenInfo>.ledgerId	LedgerId	Optional	
<TokenInfo>.expiry	Instant	Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
TokenInfoQuery query = new TokenInfoQuery()
    .setTokenId(newTokenId);

//Sign with the client operator private key, submit the query to the network and
get the token supply
long tokenSupply = query.execute(client).totalSupply;

System.out.println("The token info is " +tokenSupply);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new TokenInfoQuery()
    .setTokenId(newTokenId);

//Sign with the client operator private key, submit the query to the network and
get the token supply
const tokenSupply = (await query.execute(client)).totalSupply;

console.log("The total supply of this token is " +tokenSupply);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the query
query := hedera.NewTokenInfoQuery().
    SetTokenID(tokenId)

//Sign with the client operator private key and submit to a Hedera network
tokenInfo, err := query.Execute(client)

if err != nil {
    panic(err)
}

fmt.Printf("The token info is %v\n", tokenInfo)

//v2.1.0

{% endtab %}
{% endtabs %}
```

nft-id.md:

NFT ID

The ID of a non-fungible token (NFT). The NFT ID is composed of the token ID and a serial number.

Constructor	Description
-----	-----

| new NftId(<tokenId>,<serial>) | Initializes the NftId object |

```
java
new NftId()
```

Methods

Method	Type	Requirement
NftId.fromString(<id>)	String	Optional
NftId.fromBytes(<id>)	bytes \[]	Optional

```
{% tabs %}
{% tab title="Java" %}
java
new NftId(new TokenId(0,0,2), 56562);
```

```
// v2.0.11
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
new NftId(new TokenId(0,0,2), 56562);
```

```
// v2.0.28
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
nftId := hedera.NftID{
    TokenID: tokenId,
    SerialNumber: serialNum,
}
```

```
// v2.1.13
```

```
{% endtab %}
{% endtabs %}
```

pause-a-token.md:

Pause a token

A token pause transaction prevents the token from being involved in any kind of operation. The token's pause key is required to sign the transaction. This is a key that is specified during the creation of a token. If a token has no pause key, you will not be able to pause the token. If the pause key was not set during the creation of a token, you will not be able to update the token to add this key.

The following operations cannot be performed when a token is paused and will result in a TOKENISPAUSED status.

- Updating the token
- Transferring the token
- Transferring any other token where it has its paused key in a custom fee schedule
- Deleting the token
- Minting or burning a token
- Freezing or unfreezing an account that holds the token

Enabling or disabling KYC
Associating or disassociating a token
Wiping a token

Once a token is paused, token status will update to paused. To verify if the token's status has been updated to paused, you can request the token info via the SDK or use the token info mirror node query. If the token is not paused the token status will be unpaused. The token status for tokens that do not have an assigned pause key will state `PauseNotApplicable`.

Transaction Signing Requirements

The pause key of the token
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description	Requirement
-----	-----	-----	-----
<code>setTokenId(<tokenId>)</code>	<code>TokenId</code>	The ID of the token to pause	Required

```
{% tabs %}
{% tab title="Java" %}
java
//Create the token pause transaction and specify the token to pause
TokenPauseTransaction transaction = new TokenPauseTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the pause key, submit the
transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(pauseKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.2.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the token pause transaction, specify the token to pause, freeze the
unsigned transaction for signing
const transaction = new TokenPauseTransaction()
    .setTokenId(tokenId);
    .freezeWith(client);

//Sign with the pause key
const signTx = await transaction.sign(pauseKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);
```

```

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.2.0

{% endtab %}

{% tab title="Go" %}
go
//Create the token pause transaction, specify the token to pause, freeze the
unsigned transaction for signing
transaction, err := hedera.NewTokenPauseTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the pause key
txResponse, err = transaction.Sign(pauseKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.3.0

{% endtab %}
{% endtabs %}

# token-id.md:

Token ID

Constructs a TokenId.

| Constructor | Description |
| ----- | ----- |
| new TokenId(<shard>,<realm>,<token>) | Initializes the TokenId object |

java
new TokenId()

```

Methods

Method	Type	Description
-----	-----	-----
tokenId.fromString(<tokenId>)	String	Constructs a token ID from a String value
tokenId.fromSolidityAddress(<address>)	String	Constructs a token ID from a solidity address
tokenId.fromBytes(<bytes>)	byte[]	Constructs a token ID from bytes

```
{% tabs %}
{% tab title="Java" %}
java
tokenId tokenId = new tokenId(0,0,5);
System.out.println(tokenId);

tokenId tokenIdFromString = tokenId.fromString("0.0.3");
System.out.println(tokenIdFromString);
```

```
//Version: 2.0.1
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
const tokenId = new tokenId(0,0,5);
console.log(tokenId.toString());

const tokenIdFromString = tokenId.fromString("0.0.3");
console.log(tokenIdFromString.toString());
```

```
//Version 2.0.7
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
tokenId := hedera.TokenID {
    Shard: 0,
    Realm: 0,
    Token: 5,
}
```

```
//v2.1.0
```

```
{% endtab %}
{% endtabs %}
```

```
# token-types.md:
```

Token types

There are two types of tokens you can create using the Hedera Token Service: fungible and non-fungible tokens. A fungible (FUNGIBLECOMMON) token is a class of tokens that can be interchangeable with another in the same class. Tokens in this class share the same value and share all the same properties. A non-fungible token (NONFUNGIBLEUNIQUE) is a class of tokens that are not identical to the other tokens in the same class. This token type cannot be interchanged with other tokens and is differentiated by serial numbers that reference each unique token. The SDKs default to creating fungible tokens if the token type

during creation is not specified.

Token Type

FUNGIBLE

```
{% tabs %}
{% tab title="Java" %}
java
TokenType.FUNGIBLECOMMON
```

// v2.0.11

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
TokenType.FungibleCommon
```

// v2.0.28

```
{% endtab %}
```

```
{% tab title="Go" %}
go
hedera.TokenTypeFungibleCommon
```

// v2.1.14

```
{% endtab %}
{% endtabs %}
```

NON-FUNGIBLE

```
{% tabs %}
{% tab title="Java" %}
java
TokenType.NONFUNGIBLEUNIQUE
```

// v2.0.11

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
TokenType.NonFungibleUnique
```

// v2.0.28

```
{% endtab %}
```

```
{% tab title="Go" %}
go
hedera.TokenTypeNonFungibleUnique
```

// v2.1.14

```
{% endtab %}
{% endtabs %}
```

transfer-tokens.md:

Transfer tokens

Transfer tokens from some accounts to other accounts. The transaction must be signed by the sending account. Each negative amount is withdrawn from the corresponding account (a sender), and each positive one is added to the corresponding account (a receiver). All amounts must have a sum of zero. This does not apply to NFT token transfers. Each amount is a number with the lowest denomination possible for a token. Example: Token X has 2 decimals. Account A transfers an amount of 100 tokens by providing 10000 as the amount in the TransferList. If Account A wants to send 100.55 tokens, he must provide 10055 as the amount. If any sender account fails to have a sufficient token balance, then the entire transaction fails and none of the transfers occur, though the transaction fee is still charged. This transaction accepts zero unit token transfer operations for fungible tokens (HIP-564).

Custom Fee Tokens

Custom fee tokens are tokens that have a unique custom fee schedule associated to them. The sender account is required to pay for the custom fee(s) associated with the token that is being transferred. The sender account must have the amount of the custom fee token being transferred and the custom fee amounts to successfully process the transaction. You can check to see if the token has a custom fee schedule by requesting the token info query. Token with custom fees allow up to two levels of nesting in a transfer transaction.

```
{% hint style="warning" %}
  A max of 10 balance adjustments in its hbar transferList
  A max of 10 token fungible balance adjustments across all its
tokenTransferList's
  A max of 10 NFT ownership changes across all its tokenTransferList's
  There's also a maximum of 20 balance adjustments or NFT ownership changes
implied by a transaction (including custom fees)
  If you are transferring a token with custom fees, only two levels of nesting of
fees are allowed
  The sending account is responsible to pay for the custom token fees
{% endhint %}
```

Transaction Signing Requirements

- The key of the account sending the tokens
- The transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
addHbarTransfer(<accountId>, <value>)	AccountID, Hbar/long	Add the from and to account to transfer hbars (you will need to call this

method twice). The sending account must sign the transaction. The sender and recipient values must net zero.

```
|
| addTokenTransfer(<tokenId>, <accountId>,<value>)
| TokenId, AccountId, long
| Add the from and to account to transfer tokens (you will need to call this
method twice). The ID of the token, the account ID to transfer the tokens from
or to, and the value of the token to transfer. The sender and recipient values
must net zero. |
| addNftTransfer(<nftId>, <sender>, <receiver>)
| NftId, AccountId, AccountId | The NFT ID being transferred, the account ID
the NFT owner, the account ID of the receiver of the NFT.
|
| <p><code>addApprovedHbarTransfer(&#x3C;ownerAccountId>,&#x3C;amount>)</
code><br></p> | AccountId, Hbar
| <p>The owner account ID the spender is authorized to transfer from and the
amount.<br>Applicable to allowance transfers only.</p>
|
| <p><code>addApprovedTokenTransfer(&#x3C;tokenId>, &#x3C;accountId>,
&#x3C;value>)</code>(previewnet)<br></p> | TokenId, AccountId,
long | <p>The owner account ID and
token the spender is authorized to transfer from. The debiting account is the
owner account.<br>Applicable to allowance transfers only.<br></p>
|
| <p><code>addApprovedTokenTransferWithDecimals(&#x3C;tokenId>,
&#x3C;accountId>, &#x3C;value>, &#x3C;decimals>)</code><br></p> | TokenId,
AccountId, long, int | <p>The owner
account ID and token ID (with decimals) the spender is authorized to transfer
from. The debit account is the account ID of the sender.<br>Applicable to
allowance transfers only.</p>
|
| <p><code>addApprovedNftTransfer(&#x3C;nftId>,&#x3C;sender>,
&#x3C;receiver>)</code><br></p> | NftId,
AccountId, AccountId | <p>The NFT ID the spender is authorized to transfer. The
sender is the owner account and receiver is the receiving account.<br>Applicable
to allowance transfers only.</p>
|
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transfer transaction
TransferTransaction transaction = new TransferTransaction()
    .addTokenTransfer(tokenId, OPERATORID, -10)
    .addTokenTransfer(tokenId, accountId, 10);

//Sign with the client operator key and submit the transaction to a Hedera
network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
```

```

//Create the transfer transaction
const transaction = await new TransferTransaction()
    .addTokenTransfer(tokenId, accountId1, -10)
    .addTokenTransfer(tokenId, accountId2, 10)
    .freezeWith(client);

//Sign with the sender account private key
const signTx = await transaction.sign(accountKey1);

//Sign with the client operator private key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transfer transaction and freeze the transaction from further
modification
transaction, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId, accountId1, -10).
    AddTokenTransfer(tokenId, accountId2, 10).
    FreezeWith(client)

//Sign with the accountId1 private key, sign with the client operator key and
submit to a Hedera network
txResponse, err := transaction.Sign(accountKey1).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)
//v2.1.0

{% endtab %}
{% endtabs %}

# unfreeze-an-account.md:

Unfreeze an account

Unfreezes transfers of the specified token for the account. The transaction must
be signed by the token's Freeze Key.

```

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Unfrozen and will be able to receive or send tokens. The operation is idempotent.

Transaction Signing Requirements

Freeze key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	
-----	-----	
setTokenId(<tokenId>)	TokenId	The token for this account to unfreeze
Required		
setAccountId(<accountId>)	AccountId	The account to unfreeze
Required		

{% tabs %}

{% tab title="Java" %}

java

//Unfreeze an account

```
TokenUnfreezeTransaction transaction = new TokenUnfreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId);
```

//Freeze the unsigned transaction, sign with the sender freeze private key of the token, submit the transaction to a Hedera network

```
TransactionResponse txResponse =
transaction.freezeWith(client).sign(freezeKey).execute(client);
```

//Request the receipt of the transaction

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

//Obtain the transaction consensus status

```
Status transactionStatus = receipt8.status;
```

```
System.out.print("The transaction consensus status is " +transactionStatus);
```

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}

javascript

//Unfreeze an account and freeze the unsigned transaction for signing

```

const transaction = await new TokenUnfreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the freeze private key of the token
const signTx = await transaction.sign(freezeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt8.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Unfreeze an account and freeze the unsigned transaction for signing
transaction, err = hedera.NewUnTokenFreezeTransaction().
    SetAccountId(accountId).
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the freeze private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(freezeKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

# unpause-a-token.md:

```

Unpause a token

A token unpause transaction is a transaction that unpauses the token that was previously disabled from participating in transactions. The token's pause key is required to sign the transaction. Once the unpause transaction is submitted the token pause status is updated to unpause.

Transaction Signing Requirements:

- The pause key of the token
- Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Method	Type	Description	Requirement
-----	-----	-----	-----
setTokenId(<tokenId>)	TokenId	The ID of the token to pause	Required

Methods

```
{% tabs %}
{% tab title="Java" %}
java
//Create the token unpause transaction and specify the token to pause
TokenUnpauseTransaction transaction = new TokenUnpauseTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the pause key, submit the
transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(pauseKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.2.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the token unpause transaction, specify the token to pause, freeze the
unsigned transaction for signing
const transaction = new TokenUnpauseTransaction()
    .setTokenId(tokenId);
    .freezeWith(client);

//Sign with the pause key
const signTx = await transaction.sign(pauseKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);
```

```

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.2.0

{% endtab %}

{% tab title="Go" %}
go
//Create the token unpause transaction, specify the token to pause, freeze the
unsigned transaction for signing
transaction, err := hedera.NewTokenUnpauseTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the pause key
txResponse, err = transaction.Sign(pauseKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.3.0

{% endtab %}
{% endtabs %}

```

update-a-fee-schedule.md:

Update token custom fees

Update the custom fees for a given token. If the token does not have a fee schedule, the network response returned will be CUSTOMSCHEDULEALREADYHASNOFEES. You will need to sign the transaction with the fee schedule key to update the fee schedule for the token. If you do not have a fee schedule key set for the token, you will not be able to update the fee schedule.

Transaction Signing Requirements

- Fee schedule key
- Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Property	Description
Fee Schedule	The new fee schedule for the token

Methods

Method	Type	Requirement
setTokenId(<tokenId>)	TokenId	Required
setCustomFees(<customFees>)	List<CustomFee>	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TokenFeeScheduleUpdateTransaction transaction = new
TokenFeeScheduleUpdateTransaction()
    .setTokenId(tokenId)
    .setCustomFees(customFee)

//Freeze the unsigned transaction, sign with the fee schedule key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(feeScheduleKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//Version: 2.0.9

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze for manual signing
const transaction = await new TokenFeeScheduleUpdateTransaction()
    .setTokenId(tokenId)
    .setCustomFees(customFee)
    .freezeWith(client);

//Sign the transaction with the fee schedule key
const signTx = await transaction.sign(feeScheduleKey);

//Submit the signed transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status.toString();

console.log("The transaction consensus status is " +transactionStatus);
//Version: 2.0.26
```


Property	Description
Name	The new name of the token. The token name is specified as

a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (NUL). Is not required to be unique.

| Symbol | The new symbol of the token. The token symbol is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (NUL). Is not required to be unique.

| Treasury Account | The new treasury account of the token. If the provided treasury account is not existing or deleted, the response will be INVALIDTREASURYACCOUNTFORTOKEN. If successful, the Token balance held in the previous Treasury Account is transferred to the new one.

| Admin Key | The new admin key of the token. If the token is immutable (no Admin Key was assigned during token creation), the transaction will resolve to TOKENISIMMUTABLE. Admin keys cannot update to add new keys that were not specified during the creation of the token.

| KYC Key | The new KYC key of the token. If the token does not have currently a KYC key, the transaction will resolve to TOKENHASNOKYCKEY.

| Freeze Key | The new freeze key of the token. If the token does not have currently a freeze key, the transaction will resolve to TOKENHASNOFREEZEKEY.

| Fee Schedule Key | If set, the new key to use to update the token's custom fee schedule; if the token does not currently have this key, transaction will resolve to TOKENHASNOFEESCHEDULEKEY

| Pause Key | Update the token's existing pause key. The pause key has the ability to pause or unpause a token.

| Wipe Key | The new wipe key of the token. If the token does not have currently a wipe key, the transaction will resolve to TOKENHASNOWIPEKEY.

| Supply Key | The new supply key of the token. If the token does not have currently a supply key, the transaction will resolve to TOKENHASNOSUPPLYKEY.

| Expiration Time | The new expiry time of the token. Expiry can be updated even if the admin key is not set. If the provided expiry is earlier than the current token expiry, the transaction will resolve to INVALIDEXPIRATIONTIME.

| Auto Renew Account | The new account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval.

| Auto Renew Period | <p>The new interval at which the auto-renew account will be charged to extend the token's expiry. The default auto-renew period is 7,890,000 seconds. Currently, rent is not enforced for tokens so auto-renew payments will not be made.

NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: AUTORENEWDURATIONNOTINRANGE.</p> |

| Memo | Short publicly visible memo about the token. No guarantee of uniqueness. (100 characters max)

Transaction Signing Requirements

Admin key is required to sign to update any token properties

Updating the admin key requires the new admin key to sign

If a new treasury account is set, the new treasury key is required to sign

The account that is paying for the transaction fee

Transaction Fees

the Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type
Requirement	
-----	-----
setTokenId(<tokenId>)	TokenId
Required	
setTokenName(<name>)	String
Optional	
setTokenSymbol(<symbol>)	String
Optional	
setTreasuryAccountId(<treasury>)	AccountId Optional
setAdminKey(<key>)	Key
Optional	
setKycKey(<key>)	Key
Optional	
setFreezeKey(<key>)	Key
Optional	
setFeeScheduleKey(<key>)	Key
Optional	
setPauseKey(<key>)	Key
Optional	
setWipeKey(<key>)	Key
Optional	
setSupplyKey(<key>)	Key
Optional	
setExpirationTime(<expirationTime>)	Instant
Optional	
setTokenMemo(<memo>)	String
Optional	
setAutoRenewAccountId(<account>)	AccountId Optional
setAutoRenewPeriod(<period>)	Duration
Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TokenUpdateTransaction transaction = new TokenUpdateTransaction()
    .setTokenId(tokenId)
    .setTokenName("Your New Token Name");

//Freeze the unsigned transaction, sign with the admin private key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}
```

```

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze for manual signing
const transaction = await new TokenUpdateTransaction()
    .setTokenId(tokenId)
    .setTokenName("Your New Token Name")
    .freezeWith(client);

//Sign the transaction with the admin key
const signTx = await transaction.sign(adminKey);

//Submit the signed transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status.toString();

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze for manual signing
tokenUpdateTransaction, err := hedera.NewTokenUpdateTransaction().
    SetTokenID(tokenId).
    SetTokenName("Your New Token Name").
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin private key of the token, sign with the client operator
private key and submit the transaction to a Hedera network
txResponse, err := tokenUpdateTransaction.Sign(adminKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

# wipe-a-token.md:

```

Wipe a token

Wipes the provided amount of fungible or non-fungible tokens from the specified Hedera account. This transaction does not delete tokens from the treasury account. This transaction must be signed by the token's Wipe Key. Wiping an account's tokens burns the tokens and decreases the total supply.

If the provided account is not found, the transaction will resolve to `INVALIDACCOUNTID`.

If the provided account has been deleted, the transaction will resolve to `ACCOUNTDELETED`.

If the provided token is not found, the transaction will resolve to `INVALIDTOKENID`.

If the provided token has been deleted, the transaction will resolve to `TOKENWASDELETED`.

If an Association between the provided token and the account is not found, the transaction will resolve to `TOKENNOTASSOCIATEDTOACCOUNT`.

If Wipe Key is not present in the Token, the transaction results in `TOKENHASNOWIPEKEY`.

If the provided account is the token's Treasury Account, the transaction results in `CANNOTWIPETOKENTREASURYACCOUNT`.

On success, tokens are removed from the account and the total supply of the token is decreased by the wiped amount.

The amount provided is in the lowest denomination possible.

Example: Token A has 2 decimals. In order to wipe 100 tokens from an account, one must provide an amount of 10000. In order to wipe 100.55 tokens, one must provide an amount of 10055.

This transaction accepts zero-unit token wipe operations for fungible tokens (HIP-564)

Transaction Signing Requirements:

Wipe key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description	Requirement
<code>setTokenId(&#x3C;tokenId>)/code></code>	TokenId	The ID of the fungible or non-fungible token to remove from the account.	Required
<code>setAmount(&#x3C;amount>)/code></code>	long	Applicable to tokens of type <code>FUNGIBLECOMMON</code> . The amount of token to wipe from the specified account. The amount must be a positive non-zero number in the lowest denomination possible, not bigger than the token balance of the account.	Optional
<code>setAccount(&#x3C;accountId>)/code></code>	AccountId	The account the specified fungible or non-fungible token should be removed from.	Required
<code>setSerials(&#x3C;serials>)/code></code>	List<long>	Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code> . The list of NFTs to wipe.	Optional
<code>addSerial(&#x3C;serial>)/code></code>	long	Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code> . The NFT to wipe.	Optional

{% tabs %}

```

{% tab title="Java" %}
java
//Wipe 100 tokens from an account
TokenWipeTransaction transaction = new TokenWipeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .setAmount(100);

//Freeze the unsigned transaction, signing with the private key of the payer and
the token's wipe key; submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(accountKey).sign(wipeKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Wipe 100 tokens from an account and freeze the unsigned transaction for manual
signing
const transaction = await new TokenWipeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .setAmount(100)
    .freezeWith(client);

//Sign with the payer account private key, sign with the wipe private key of the
token
const signTx = await (await transaction.sign(accountKey)).sign(wipeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());

{% endtab %}

{% tab title="Go" %}
go
//Wipe 100 tokens and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenBurnTransaction().
    SetAccountId(accountId).
    SetTokenID(tokenId).
    SetAmount(1000).
    FreezeWith(client)

if err != nil {
    panic(err)
}

```

```

//Sign with the payer account private key, sign with the wipe private key of the
token
txResponse, err := transaction.Sign(accountKey).Sign(wipeKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

create-a-schedule-transaction.md:

Create a scheduled transaction

```

{% hint style="info" %}
TokenTransfer, ConsensusSubmitMessage, CryptoApproveAllowance, TokenMint and
TokenBurn transactions are the transaction types that can be scheduled.
Additional schedulable transactions will be added in future releases.
{% endhint %}

```

A transaction that creates a schedule entity on a Hedera network. The entity ID for a schedule transaction is referred to as the ScheduleID. After successfully executing a schedule create transaction, you can obtain the ScheduleID by requesting the receipt of the transaction immediately after the transaction was executed. The receipt also contains the scheduled transaction ID. The scheduled transaction ID is used to request the record of the scheduled transaction if it is successfully executed.

When creating a transaction to schedule you do not need to use `.freezeWith(client)` method.

Example:

```

java
TransferTransaction transactionToSchedule = new TransferTransaction()
    .addHbarTransfer(newAccountId, Hbar.fromTinybars(-1))
    .addHbarTransfer(myAccountId, Hbar.fromTinybars(1));

```

Schedule Transaction Duplicate

If two users submit the same schedule create transaction, the first one to reach consensus will create the schedule ID and the second one will have the schedule ID returned in the receipt of the transaction. The receipt status of the second identical schedule transaction will return a "IDENTICALSCHEDULEALREADYCREATED" response from the network. The user who submits the second transaction would need to submit a ScheduleSign transaction to add their signature to the schedule

transaction.

Schedule Transaction Deletion

To retain the ability to delete a schedule transaction, you will need to populate the admin key field when creating a schedule transaction. The admin key will be required to sign the ScheduleDelete transaction to delete the scheduled transaction from the network. If you do not assign an admin key during the creation of the schedule transaction, you will have an immutable schedule transaction.

Transaction Signing Requirements

The key of the account paying for the creation of the schedule transaction
The key of the payer account ID paying for the execution of the scheduled transaction. If the payer account is not specified, the operator account will be used to pay for the execution by default.

The admin key if set

You can optionally sign with any of the required signatures for the scheduled (inner) transaction. Freeze the schedule transaction and call the .sign() method to add signatures.

Transaction Properties

{% hint style="info" %}

Note: If you do not set the payer account ID the schedule transaction is immutable.

{% endhint %}

Field	Description
Schedulable Transaction Body	The transaction body of the transaction that is being scheduled
Admin Key	A key that can delete the schedule transaction prior to execution or expiration
Payer Account ID	The account which is going to pay for the execution of the scheduled transaction. If not populated, the scheduling account is charged (optional).
Memo	Publicly visible information about the schedule entity, up to 100 bytes. No guarantee of uniqueness (optional).

Constructor	Description
new ScheduleCreateTransaction()	Initializes the ScheduleCreateTransaction object

```
java
new ScheduleCreateTransaction()
```

Methods

Method	Type	Requirement
setScheduledTransaction(transaction)	Transaction	Required

<code>setAdminKey(&#x3C;key>)</code>	Key	Optional
<code>setPayerAccountId(&#x3C;id>)</code>	AccountId	Optional
<code>setScheduleMemo(&#x3C;memo>)</code>	String	Optional
<code>setExpirationTime(expirationTime)</code>	Instant	Optional
<code>setWaitForExpiry(waitForExpiry)</code>	boolean	Optional
<code>getAdminKey()</code>	Key	Optional
<code>getPayerAccountId()</code>	AccountId	Optional
<code>getScheduleMemo()</code>	String	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create a schedule transaction
ScheduleCreateTransaction transaction = new ScheduleCreateTransaction()
    .setScheduledTransaction(transactionToSchedule);

//Sign with the client operator key and submit the transaction to a Hedera
network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the schedule ID
ScheduleId scheduleId = receipt.scheduleId;
System.out.println("The schedule ID of the schedule transaction is "
+scheduleId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a schedule transaction
const transaction = new ScheduleCreateTransaction()
    .setScheduledTransaction(transactionToSchedule);

//Sign with the client operator key and submit the transaction to a Hedera
network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the schedule ID
const scheduleId = receipt.scheduleId;
console.log("The schedule ID of the schedule transaction is " +scheduleId);

{% endtab %}

{% tab title="Go" %}
go
    //Create a schedule transaction
    transaction, err := transactionToSchedule.Schedule()

    if err != nil {
        panic(err)
    }

//Sign with the client operator key and submit the transaction to a Hedera
network
txResponse, err := transaction.Execute(client)
```

```

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the schedule ID from the receipt
scheduleId := receipt.ScheduleID

fmt.Printf("The new token ID is %v\n", scheduleId)

{% endtab %}
{% endtabs %}

```

delete-a-schedule-transaction.md:

Delete a scheduled transaction

A transaction that deletes a scheduled transaction from the network. You can delete a scheduled transaction if only the admin key was set during the creation of the scheduled transaction. If an admin key was not set, the attempted deletion will result in "SCHEDULEISIMMUTABLE" response from the network. Once the scheduled transaction is deleted, the scheduled transaction will be marked as deleted with the consensus timestamp the scheduled transaction was deleted at.

Transaction Signing Requirements

The admin key of the scheduled transaction

Transaction Properties

Field	Description
Schedule ID	The ID of the scheduled transaction

Methods

Method	Type	Requirement
setScheduleId(scheduleId)	ScheduleId	Required
getScheduleId()	ScheduleId	Optional

```

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction and sign with the admin key
ScheduleDeleteTransaction transaction = new ScheduleDeleteTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(adminKey);

//Sign with the operator key and submit to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Get the transaction receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

```

```

//Get the transaction status
Status transactionStatus = receipt.status;
System.out.println("The transaction consensus status is " +transactionStatus);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and sign with the admin key
const transaction = await new ScheduleDeleteTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(adminKey);

//Sign with the operator key and submit to a Hedera network
const txResponse = await transaction.execute(client);

//Get the transaction receipt
const receipt = await txResponse.getReceipt(client);

//Get the transaction status
const transactionStatus = receipt.status;
console.log("The transaction consensus status is " +transactionStatus);

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction
transaction, err := hedera.NewScheduleDeleteTransaction()
    SetScheduleID(scheduleId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin key, sign with the client operator private key and submit
the transaction to a Hedera network
txResponse, err := transaction.Sign(adminKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status:= receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

{% endtab %}
{% endtabs %}

# get-schedule-info.md:

Get schedule info

```

A query that returns information about the current state of a schedule transaction on a Hedera network.

Schedule Info Response

Field	Description

Schedule ID	The ID of the schedule transaction
Creator Account ID	The Hedera account that created the schedule transaction in x.y.z format
Payer Account ID	The Hedera account paying for the execution of the schedule transaction in x.y.z format
Scheduled Transaction Body	The scheduled transaction (inner transaction).
Signatories	The signatories that have provided signatures so far for the schedule transaction
Admin Key	The Key which is able to delete the schedule transaction if set
Expiration Time	The date and time the schedule transaction will expire
Executed Time	The time the schedule transaction was executed. If the schedule transaction has not executed this field will be left null.
Deletion Time	The consensus time the schedule transaction was deleted. If the schedule transaction was not deleted, this field will be left null.
Memo	Publicly visible information about the Schedule entity, up to 100 bytes. No guarantee of uniqueness.

Query Signing Requirements

The client operator key is required to sign the query request

Methods

Method	Type	Requirement
-----	-----	-----
setScheduleId(<scheduleId>)	ScheduleId	Required
<ScheduleInfo>.scheduleId	ScheduleId	Optional
<ScheduleInfo>.scheduledTransactionId	TransactionId	Optional
<ScheduleInfo>.creatorAccountId	AccountId	Optional
<ScheduleInfo>.payerAccountId	AccountId	Optional
<ScheduleInfo>.adminKey	Key	Optional
<ScheduleInfo>.signatories	Key	Optional
<ScheduleInfo>.deletedAt	Instant	Optional
<ScheduleInfo>.expirationAt	Instant	Optional
<ScheduleInfo>.memo	String	Optional
<ScheduleInfo>.waitForExpiry	boolean	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
```

```

ScheduleInfoQuery query = new ScheduleInfoQuery()
    .setScheduleId(scheduleId);

//Sign with the client operator private key and submit the query request to a
node in a Hedera network
ScheduleInfo info = query.execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new ScheduleInfoQuery()
    .setScheduleId(scheduleId);

//Sign with the client operator private key and submit the query request to a
node in a Hedera network
const info = await query.execute(client);

{% endtab %}

{% tab title="Go" %}
go
//Create the query
query := hedera.NewScheduleInfoQuery().
    SetScheduleID(scheduleId)

//Sign with the client operator private key and submit to a Hedera network
scheduleInfo, err := query.Execute(client)

if err != nil {
    panic(err)
}

{% endtab %}
{% endtabs %}

```

network-response-messages.md:

Network Response Messages

Network Response	Description
-----	-----
INVALIDSCHEDULEID	The Scheduled entity does not exist; or has now expired, been deleted, or been executed
SCHEDULEISIMMUTABLE	The Scheduled entity cannot be modified. Admin key was not set during the creation of the Scheduled entity.
INVALIDSCHEDULEPAYERID	The provided Scheduled Payer does not exist
INVALIDSCHEDULEACCOUNTID	The Schedule Create Transaction TransactionID account does not exist
NEWVALIDSIGNATURES	The provided sig map did not contain any new valid signatures from required signers of the scheduled transaction
UNRESOLVABLEREQUIREDSIGNERS	The required signers for a scheduled transaction cannot be resolved, for example because they

do not exist or have been deleted		
UNPARSEABLESCHEDULEDTRANSACTION		The bytes
allegedly representing a transaction to be scheduled could not be parsed		
UNSCHEDULABLETRANSACTION		ScheduleCreate
and ScheduleSign transactions cannot be scheduled		
SOMESIGNATURESWEREINVALID		At least one of the
signatures in the provided sig map did not represent a valid signature for any		
required signer		
TRANSACTIONIDFIELDNOTALLOWED		The scheduled and
nonce fields in the TransactionID may not be set in a top-level transaction		
IDENTICALSCHEDULEALREADYCREATED		A schedule already
exists with the same identifying fields of an attempted ScheduleCreate (that is,		
all fields other than scheduledPayerAccountID)		
SCHEDULEALREADYDELETED		A schedule being
signed or deleted has already been deleted		
SCHEDULEPENDINGEXPIRATION		A schedule being
signed or deleted has passed it's expiration date and is pending execution if		
needed and then expiration		
SCHEDULEFUTUREGASLIMITEXCEEDED		The scheduled
transaction could not be created because it would cause the gas limit to be		
violated on the specified expiration time		
SCHEDULEFUTURETHROTTLEEXCEEDED		The scheduled
transaction could not be created because it would cause throttles to be violated		
on the specified expiration time		
SCHEDULEEXPIRATIONTIMEMUSTBEHIGHERTHANCONSENSUSTIME		The scheduled
transaction could not be created because it's expiration\time was less than or		
equal to the consensus time		
SCHEDULEEXPIRATIONTIMETOOFARINFUTURE		The scheduled
transaction could not be created because it's expiration time was too far in the		
future		

README.md:

Schedule Transaction

schedule-id.md:

Schedule ID

The entity ID of a schedule transaction.

A ScheduleId is composed of a \<shardNum>.\<realmNum>.\<scheduleNum> (eg. 0.0.10).

shardNum represents the shard number (shardId). It will default to 0 today, as Hedera only performs in one shard.

realmNum represents the realm number (realmId). It will default to 0 today, as realms are not yet supported.

scheduleNum represents the schedule number (scheduleId)

Together these values make up your ScheduleId. When a ScheduleId is requested in a field, be sure enter all three values.

Constructor		Type		Description
-----		:-----		

```

----- |
| new ScheduleId(<shardNum>,<realmNum>,<scheduleNum>) | long, long, long |
Constructs a ScheduleId with 0 for shardNum and realmNum (e.g.,
0.0.<scheduleNum>) |

```

Example

```

{% tabs %}
{% tab title="Java" %}
java
ScheduleId scheduleID = new ScheduleId(0,0,10);
System.out.println(scheduleID)

```

```

{% endtab %}

```

```

{% tab title="JavaScript" %}
javascript
const scheduleID = new ScheduleId(0,0,10);
console.log(scheduleID)

```

```

{% endtab %}
{% endtabs %}

```

sign-a-schedule-transaction.md:

Sign a scheduled transaction

A transaction that appends signatures to a scheduled transaction. You will need to know the schedule ID to reference the scheduled transaction to submit signatures. A record will be generated for each ScheduleSign transaction that is successful and the scheduled entity will subsequently update with the public keys that have signed the scheduled transaction. To view the keys that have signed the scheduled transaction, you can query the network for the schedule info. Once a scheduled transaction receives the last required signature, the scheduled transaction executes.

Transaction Signing Requirements

The key of the account paying for the transaction

Transaction Properties

Field	Description
-----	-----
Schedule ID	The ID of the scheduled transaction to submit the signature for

Methods

```

<table><thead><tr><th
width="351.3333333333333">Method</th><th>Type</th><th>Requirement</th></tr></thead><tbody><tr><td><code>setScheduleId(&#x3C;scheduleId>)</code></td><td>ScheduleId</td><td>Required</td></tr><tr><td><code>clearScheduleId(&#x3C;scheduleId>)</code></td><td>ScheduleId</td><td>Optional</td></tr><tr><td><code>getScheduleId()</code></td><td>ScheduleId</td><td>Optional</td></tr></tbody></table>

```

```

{% tabs %}
{% tab title="Java" %}
{% code title="Java" %}
java

```

```

//Create the transaction
ScheduleSignTransaction transaction = new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(privateKeySigner1);

//Sign with the client operator key to pay for the transaction and submit to a
Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction status
Status transactionStatus = receipt.status;
System.out.println("The transaction consensus status is " +transactionStatus);

{% endcode %}
{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(privateKeySigner1);

//Sign with the client operator key to pay for the transaction and submit to a
Hedera network
const txResponse = await transaction.execute(client);

//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction status
const transactionStatus = receipt.status;
console.log("The transaction consensus status is " +transactionStatus);

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction
transaction, err := hedera.NewScheduleSignTransaction().
    SetScheduleID(scheduleId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with one of the required signatures, sign with the client operator
private key and submit the transaction to a Hedera network
txResponse, err := transaction.Sign(privateKeySigner1).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)

if err != nil {

```



```

    panic(err)
}

//Get the transaction consensus status
status:= receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

{% endtab %}
{% endtabs %}

# local-provider.md:

Local Provider

{% hint style="info" %}
This feature is available in the Hedera JavaScript SDK only. (version >=2.14.0).
{% endhint %}

LocalProvider is a quality of life implementation that creates a provider using
the HEDERANETWORK environment variable.

The LocalProvider() requires the following variable to be defined in the .env
file. The .env file is located in the root directory of the project.

    HEDERANETWORK
    The network the wallet submits transactions to

{% code title=".env" %}

//Example .env file
HEDERANETWORK= previewnet/testnet/mainnet (select one network)

{% endcode %}

class LocalProvider implements Wallet

Constructor

new <mark style="color:purple;">LocalProvider</mark><mark
style="color:purple;">()</mark>

Instantiates the LocalProvider object. The local provider is built using
HEDERANETWORK network specified in the .env file.

Methods

<LocalProvider>.<mark style="color:purple;">getAccountBalance()</mark><mark
style="color:purple;">\\</mark> -> Promise \\<AccountBalance>\\

Returns the account balance of the account in the local wallet.

<LocalProvider>.<mark style="color:purple;">getAccountInfo()</mark><mark
style="color:purple;">\\</mark> -> Promise \\<AccountInfo>\\

Returns the account information of the account in the local wallet.

<LocalProvider>.<mark style="color:purple;">getAccountRecords()</mark><mark
style="color:purple;">\\</mark> -> Promise \\<AccountInfo>\\

Returns the last transaction records for this account using
TransactionRecordQuery.

```

```
<LocalProvider>.<mark style="color:purple;">getLedgerId()</mark><mark style="color:purple;">-></mark>\\ LedgerId\\
```

Returns the ledger ID (previewnet, testnet, or mainnet).

```
<LocalProvider>.<mark style="color:purple;">getMirrorNetwork()</mark><mark style="color:purple;">-></mark>\\ string\\
```

The mirror network the wallet is connected to.

```
<LocalProvider>.<mark style="color:purple;">getNetwork()</mark><mark style="color:purple;">-></mark>\\ \[key: string]: string | <mark style="color:purple;">\\\\</mark> AccountId\\
```

Returns the network map information.

```
<LocalProvider>.<mark style="color:purple;">getTransactionReceipt()</mark><mark style="color:purple;">-></mark> Promise<TransactionReceipt>
```

Returns the transaction receipt.

```
<LocalProvider>.<mark style="color:purple;">waitForReceipt()</mark><mark style="color:purple;">-></mark> Promise<TransactionReceipt>
```

Wait for the receipt for a transaction response.

```
<LocalProvider>.<mark style="color:purple;">call(</mark><RequestT, ResponseT, OutputT>(request: Executable<RequestT, ResponseT, OutputT><mark style="color:purple;"></mark><mark style="color:purple;">-></mark>\\ Promise<Output>\\
```

Sign and send a request using the wallet.

Local Wallet Example:

```
javascript
require("dotenv").config();
import { Wallet, LocalProvider } from "@hashgraph/sdk";

async function main() {
    const wallet = new Wallet(
        process.env.OPERATORID,
        process.env.OPERATORKEY,
        new LocalProvider()
    );

    const info = await wallet.getAccountInfo();

    console.log(info.key                                = ${info.key.toString()});
}

void main();
```

Sign with Local Wallet Example:

```
javascript
require("dotenv").config();
const {Wallet, LocalProvider, TransferTransaction} = require("@hashgraph/sdk");

async function main() {
    const wallet = new Wallet(
```

```

        process.env.OPERATORID,
        process.env.OPERATORKEY,
        new LocalProvider()
    );

    //Transfer 1 hbar from my wallet account to account 0.0.3
    const transaction = await new TransferTransaction()
        .addHbarTransfer(wallet.getAccountId(), -1)
        .addHbarTransfer("0.0.3", 1)
        .freezeWithSigner(wallet);

    //Sign the transaction
    const signTransaction = await transaction.signWithSigner(wallet);

    //Submit the transaction
    const response = await signTransaction.executeWithSigner(wallet);

    //Get the receipt
    const receipt = await wallet.getProvider().waitForReceipt(response);

    console.log(status: ${receipt.status.toString()});
}

main();

```

provider.md:

Provider

```

{% hint style="info" %}
This feature is available in the Hedera JavaScript SDK only. (version >=2.14.0).
{% endhint %}

```

Provider provides access to a Hedera network to which requests should be submitted to. The Hedera network can be specified to previewnet, testnet, or mainnet.

The most important method on the provider interface is the call method which allows a user to submit any request and get the correct response for that request. For instance, Java

```

javascript
// Balance of node account ID 0.0.3
const balance = provider.call(new
AccountBalanceQuery().setAccountId(AccountId.fromString("0.0.3")));
Interface
Provider

```

Methods

```

<mark style="color:purple;">getLedgerId()</mark><mark style="color:purple;">-
></mark>\\ LedgerId\\

```

Returns the ledger ID of the current network.

```

<mark style="color:purple;">getNetwork()</mark><mark
style="color:purple;">-></mark>\\ Map < [key: string]: (string | AccountId)>\\

```

Returns the entire network map for the current network.

```

<mark style="color:purple;">getAccountBalance(</mark>accountId: AccountId |
string<mark style="color:purple;"></mark><mark

```

```
style="color:purple;">-></mark>\\ Promise<AccountBalance>\\
```

Get the balance for the specified account ID.

```
<mark style="color:purple;">getAccountInfo(</mark>accountId: AccountId |
string<mark style="color:purple;">)</mark><mark
style="color:purple;">-></mark>\\ Promise<AccountInfo>\\
```

Get the info for the specified account ID.

```
<mark style="color:purple;">getAccountRecords(</mark>accountId: AccountId |
string<mark style="color:purple;">)</mark><mark
style="color:purple;">-></mark>\\ Promise<TransactionRecord[]>\\
```

Get the account record for the specified account ID.

```
<mark
style="color:purple;">getTransactionReceipt(</mark>transactionId:TransactionId<m
ark style="color:purple;">)</mark>: TransactionReceipt
```

Get a receipt for the specified transaction ID.

```
<mark style="color:purple;">waitForReceipt(</mark>response:
TransactionResponse<mark style="color:purple;">)</mark><mark
style="color:purple;">-></mark>\\ Promise<TransactionReceipt>\\
```

Execute multiple TransactionReceiptQuery's until we get an erroring status code, or a success state code.

```
<mark style="color:purple;">call(</mark><RequestT, ResponseT, OutputT>(request:
Executable<RequestT, ResponseT, OutputT><mark
style="color:purple;">)</mark><mark style="color:purple;">-></mark>\\ Promise
<Output>\\
```

Responsible for serializing your request and sending it over the wire to be executed, and then deserializing the response into the appropriate type.

README.md:

Signature Provider

signer.md:

Signer

```
{% hint style="info" %}
This feature is available in the Hedera JavaScript SDK only. (version >=2.14.0).
{% endhint %}
```

The Signer class an interface and is responsible for signing requests.

Interface Signer

Methods

```
<mark style="color:purple;">getLedgerId(</mark><mark style="color:purple;">-
></mark>\\ LedgerId\\
```

Returns the ledger ID.

```
<mark style="color:purple;">getAccountId(</mark><mark style="color:purple;">-
```

></mark>AccountId

Returns the account ID associated with this signer.

<mark style="color:purple;">getAccountKey()</mark><mark style="color:purple;">-></mark>Key

Returns the account key.

<mark style="color:purple;">getNetwork()</mark><mark style="color:purple;">-></mark>\[Key:string]: string

Returns the account key.

<mark style="color:purple;">sign (</mark>messages; Uint8Array[]<mark style="color:purple;">)</mark><mark style="color:purple;">-></mark>Promise<SignerSignature[]>

Sign an arbitrary list of messages.

<mark style="color:purple;">getAccountBalance()</mark><mark style="color:purple;">-></mark>\ Promise<AccountBalance>\

Returns the account balance.

<mark style="color:purple;">getAccountInfo</mark><mark style="color:purple;">()</mark>-></mark>\ Promise<AccountInfo>\

Returns the account info.

<mark style="color:purple;">getAccountRecords</mark><mark style="color:purple;">()</mark>-></mark>\ Promise<TransactionRecord[]>\

Fetch the account record for the signer's account ID.

<mark style="color:purple;">signTransaction \<T extends Transaction></mark>(transaction:T)->\ Promise<T>\

Signs a transaction, returning the signed transaction

NOTE: This method is allowed to mutate the parameter being passed in so the returned transaction is not guaranteed to be a new instance of the transaction.

<mark style="color:purple;">checkTransaction<T extends Transaction></mark><mark style="color:purple;">\</mark> (transaction: T)-> Promise<T>\

Check whether all the required fields are set appropriately. Fields such as the transaction ID's account ID should either be null or be equal to the signer's account ID, and the node account IDs on the request should exist within the signer's network.

<mark style="color:purple;">populateTransaction<T extends Transaction></mark><mark style="color:purple;">\</mark> (transaction: T)-> Promise<T>\

Populate the requests with the required fields. The transaction ID should be constructed from the signer's account ID, and the node account IDs should be set using the signer's network.

<mark style="color:purple;">call(</mark><RequestT, ResponseT, OutputT>(request: Executable<RequestT, ResponseT, OutputT><mark style="color:purple;">)</mark><mark style="color:purple;">-></mark>\ Promise<Output>\

Responsible for serializing your request and sending it over the wire to be executed, and then deserializing the response into the appropriate type.

Note: This is a wrapper around the `Provider.call()` method.

wallet.md:

Wallet

```
{% hint style="info" %}
```

This feature is available in the Hedera JavaScript SDK only. (version >=2.14.0).

```
{% endhint %}
```

The Wallet extends the Signer. The Signer is responsible for Signing requests while the Provider is responsible for communication between an application and a Hedera network, but is not required to communicate directly with a Hedera network. Note this means the Provider can for instance communicate with some third party service which finally communicates with a Hedera network.

class Wallet extends Signer

Constructor

```
new<mark style="color:purple;">wallet(</mark><accountId>, <privateKey>,<br><provider><mark style="color:purple;"></mark></mark>
```

Constructs a local wallet from an account ID, private key, and provider.

Methods

```
<mark style="color:purple;"><Wallet>.getLedgerId()</mark><mark<br>style="color:purple;">-></mark>\\ LedgerId\\
```

Returns the ledger ID.

```
<mark style="color:purple;"><Wallet>.getAccountId()</mark><mark<br>style="color:purple;">-></mark>AccountId
```

Returns the account ID associated with this signer.

```
<mark style="color:purple;"><Wallet>.getAccountKey()</mark><mark<br>style="color:purple;">-></mark>Key
```

Returns the account key.

```
<mark style="color:purple;"><Wallet>.getNetwork()</mark><mark<br>style="color:purple;">-></mark>\\[Key:string]: string
```

Returns the full consensus network being used.

```
<mark style="color:purple;"><Wallet>.getMirrorNetwork()</mark><mark<br>style="color:purple;">-></mark>string\\[
```

Returns the mirror node for the network.

```
<mark style="color:purple;"><Wallet>.sign (</mark>messages; Uint8Array[]<mark<br>style="color:purple;">)</mark><mark style="color:purple;">-></mark><br>Promise<SignerSignature[]>
```

Sign an arbitrary list of messages.

```
<mark style="color:purple;"><Wallet>.getAccountBalance()</mark><mark<br>style="color:purple;">-></mark>\\ Promise<AccountBalance>\\
```

Returns the account balance.

```
<mark style="color:purple;"><Wallet>.getAccountInfo</mark><mark style="color:purple;">()-></mark>\\ Promise<AccountInfo>\\
```

Returns the account info.

```
<mark style="color:purple;"><Wallet>.getAccountRecords</mark><mark style="color:purple;">()-></mark>\\ Promise<TransactionRecord[]>\\
```

Fetch the last transaction records for this account using TransactionRecordQuery.

```
<mark style="color:purple;"><Wallet>.getProvider</mark><mark style="color:purple;">()-></mark>\\ Provider\\
```

Returns the Hedera network.

```
<mark style="color:purple;"><Wallet>.</mark><mark style="color:purple;">signTransaction \<T extends Transaction></mark>(transaction:T )->\\ Promise<T>\\
```

Signs a transaction, returning the signed transaction

NOTE: This method is allowed to mutate the parameter being passed in so the returned transaction is not guaranteed to be a new instance of the transaction.

```
<mark style="color:purple;"><Wallet>.checkTransaction<T extends Transaction></mark><mark style="color:purple;">\\</mark> ( transaction: T )-> Promise<T>\\
```

Check whether all the required fields are set appropriately. Fields such as the transaction ID's account ID should either be null or be equal to the signer's account ID, and the node account IDs on the request should exist within the signer's network.

```
<mark style="color:purple;"><Wallet>.populateTransaction<T extends Transaction></mark><mark style="color:purple;">\\</mark> ( transaction: T )-> Promise<T>\\
```

Populate the requests with the required fields. The transaction ID should be constructed from the signer's account ID, and the node account IDs should be set using the signer's network.

```
<mark style="color:purple;"><Wallet>.call(</mark><RequestT, ResponseT, OutputT>(request: Executable<RequestT, ResponseT, OutputT><mark style="color:purple;">)</mark><mark style="color:purple;">-></mark>\\ Promise<Output>\\
```

Responsible for serializing your request and sending it over the wire to be executed, and then deserializing the response into the appropriate type.

Note: This is a wrapper around the Provider.call() method.

call-a-smart-contract-function.md:

Call a smart contract function

The transaction calls a function of the given smart contract instance, giving it functionParameters as its input. The call can use at maximum the given amount of gas - the paying account will not be charged for any unspent gas.

If this function results in data being stored, an amount of gas is calculated that reflects this storage burden.

The amount of gas used, as well as other attributes of the transaction, e.g. size, and number of signatures to be verified, determine the fee for the transaction – which is charged to the paying account.

Transaction Signing Requirements

The key of the transaction fee-paying account

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Method	Type	Description	Requirement
<code>setContractId(&#x3C;contractId>)</code>	ContractID	The ID of the contract to call.	Required
<code>setFunction(&#x3C;name, params>)</code>	String, ContractFunctionParameters	Which function to call and the parameters to pass to the function.	Required
<code>setGas(&#x3C;gas>)</code>	long	The maximum amount of gas to use for the call.	Required
<code>setPayableAmount(&#x3C;amount>)</code>	Hbar	Number of HBARS sent (the function must be payable if this is nonzero)	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
ContractCreateTransaction transaction = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(1000000000)
    .setFunction("setMessage", new ContractFunctionParameters()
        .addString("hello from hedera again!"))

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(1000000000)
    .setFunction("setMessage", new ContractFunctionParameters()
        .addString("hello from hedera again!"))
```



```

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction, err := hedera.NewContractExecuteTransaction().
    SetContractID(newContractID).
    SetGas(1000000000).
    SetFunction("setMessage", contractFunctionParams)

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
txResponse, err := transaction.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status %v\n", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

```

create-a-smart-contract.md:

Create a smart contract

A transaction that creates a new smart contract instance. After the contract is created you can get the new contract ID by requesting the receipt of the transaction. To create the solidity smart contract, you can use remix or another Solidity compiler. After you have the hex-encoded bytecode of the smart contract you need to store that on a file using the Hedera File Service. Then you will create the smart contract instance that will run the bytecode stored in the Hedera file, referenced by file ID. Alternatively, you can use the `ContractCreateFlow()` API to create the file storing the bytecode and contract in a single step.

The constructor will be executed using the given amount of gas, and any unspent gas will be refunded to the paying account. Constructor inputs are passed in the

constructorParameters.

If this constructor stores information, it is charged gas to store it. There is a fee in hbars to maintain that storage until the expiration time, and that fee is added as part of the transaction fee.

{% hint style="info" %}
Solidity Support

The latest version of Solidity is supported on all networks (v0.8.9).
{% endhint %}

{% hint style="info" %}
Smart Contract State Size and Gas Limits

The max contract key value pairs are 16,384,000 (\100 MB). The system gas throttle is 15 million gas per second. Contract call and contract create are throttled at 8 million gas per second.
{% endhint %}

{% hint style="info" %}
Admin Key Support for Contracts

With HIP-904, contracts now support Admin Key settings. Set an Admin Key during the ContractCreate transaction to manage and update token properties. For frictionless-airdrop enabled contracts, set maxAutoAssociations during the ContractCreate transaction, ensuring that balance and associations can be managed with a valid Admin Key.
{% endhint %}

Transaction Signing Requirements

The client operator account is required to sign the transaction.
The admin key, if specified.
The key of the autorenewal account, if specified.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate the cost of your transaction fee.

Smart Contract Properties

Field	Description
-----	-----
-----	-----
-----	-----
-----	-----
-----	-----
-----	-----
Admin Key	Sets the state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
Gas	The gas to run the constructor.
Initial Balance	The initial number of hbars to put into the cryptocurrency account associated with and owned by the smart contract.

Byte Code File	The file containing the hex encoded smart contract byte code.
Staked ID	<p>The node account or node ID to which this contract account is staking to. See HIP-406.</p> <p>Note: Accounts cannot stake to contract accounts. This will fixed in a future release.</p>
Decline Rewards	Some contracts may choose to stake their hbars and decline receiving rewards. If set to true, the contract account will not earn rewards when staked. The default value is false. See HIP-406.
Auto Renew Account ID	An account to charge for auto-renewal of this contract. If not set, or set to an account with zero hbar balance, the contract's own hbar balance will be used to cover auto-renewal fees.
Auto Renew Period	The period that the instance will charge its account every this many seconds to renew.
Automatic Token Associations	The maximum number of tokens that this contract can be automatically associated with (i.e., receive air-drops from).
Constructor Parameters	The constructor parameters to pass.
Memo (max 100 bytes)	The memo to be associated with this contract.

ContractCreateFlow()

The `ContractCreateFlow()` streamlines the creation of a contract by taking the bytecode of the contract and creating the file on Hedera to store the bytecode for you.

First, a `FileCreateTransaction()` will be executed to create a file on Hedera to store the specified contract bytecode. Then, the `ContractCreateTransaction()` will be executed to create the contract instance on Hedera. Lastly, a `FileDeleteTransaction()` will be executed to remove the file.

The response will return the contract create transaction information like the new contract ID. You will not get the ID of the file that was created that stored your contract bytecode. If you would like to know the file ID of your contract bytecode, you can create a file and use the `ContractCreateTransaction()` API directly.

Methods

```
{% hint style="info" %}
```

Note: Please refer to `ContractCreateTransaction()` for a complete list of applicable methods.

```
{% endhint %}
```

Method	Type
-----	-----
setBytecode(<bytecode>)	byte
setBytecode(<bytecode>)	String
setBytecode(<bytecode>)	ByteString
setMaxChunks(<maxChunks>)	int

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```

java
//Create the transaction
ContractCreateFlow contractCreate = new ContractCreateFlow()
    .setBytecode(bytecode)
    .setGas(1000000);

//Sign the transaction with the client operator key and submit to a Hedera
network
TransactionResponse txResponse = contractCreate.execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the new contract ID
ContractId newContractId = receipt.contractId;

System.out.println("The new contract ID is " +newContractId);
//SDK Version: v2.10.0-beta.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const contractCreate = new ContractCreateFlow()
    .setGas(1000000)
    .setBytecode(bytecode);

//Sign the transaction with the client operator key and submit to a Hedera
network
const txResponse = contractCreate.execute(client);

//Get the receipt of the transaction
const receipt = (await txResponse).getReceipt(client);

//Get the new contract ID
const newContractId = (await receipt).contractId;

console.log("The new contract ID is " +newContractId);
//SDK Version: v2.11.0-beta.1

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
contractCreate := hedera.NewContractCreateFlow().
    SetGas(1000000).
    SetBytecode(byteCode)

//Sign the transaction with the client operator key and submit to a Hedera
network
txResponse, err := contractCreate.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the topic ID

```

```
newContractId := receipt.ContractID
```

```
fmt.Printf("The new topic ID is %v\n", newContractId)
//SDK Version: 2.11.0-beta.1
```

```
{% endtab %}
{% endtabs %}
```

```
ContractCreateTransaction()
```

Creates a smart contract instance using the file ID of the contract bytecode.

Constructor	Description
new ContractCreateTransaction()	Initializes the ContractCreateTransaction() object

Methods

Method	Type
Required	
setGas(<gas>)	long
Required	
setBytecodeFileId(<fileId>)	FileId Required
setInitialBalance(<initialBalance>)	Hbar
Optional	
setAdminKey(<keys>)	Key
Optional	
setConstructorParameters(<constructorParameters>)	byte \[]
Optional	
setConstructorParameters(<constructorParameters>)	ContractFunctionParameters
Optional	
setContractMemo(<memo>)	String
Optional	
setStakedNodeId(<stakedNodeId>)	long
Optional	
setStakedAccountId(<stakedAccountId>)	AccountId
Optional	
setDeclineStakingReward(<declineStakingReward>)	boolean
Optional	
setAutoRenewAccountId(<accountId>)	AccountId
Optional	
setAutoRenewPeriod(<autoRenewPeriod>)	Duration
Optional	
setMaxAutomaticTokenAssociations()	int
Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
ContractCreateTransaction transaction = new ContractCreateTransaction()
    .setGas(100000)
    .setBytecodeFileId(bytecodeFileId);
```

```
//Modify the default max transaction fee (default: 1 hbar)
ContractCreateTransaction modifyTransactionFee =
transaction.setMaxTransactionFee(new Hbar(16));
```

```
//Sign the transaction with the client operator key and submit to a Hedera
```

```

network
TransactionResponse txResponse = modifyTransactionFee.execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the new contract ID
ContractId newContractId = receipt.contractId;

System.out.println("The new contract ID is " +newContractId);
//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new ContractCreateTransaction()
    .setGas(1000000)
    .setBytecodeFileId(bytecodeFileId);

//Modify the default max transaction fee (default: 1 hbar)
const modifyTransactionFee = transaction.setMaxTransactionFee(new Hbar(16));

//Sign the transaction with the client operator key and submit to a Hedera
network
const txResponse = await modifyTransactionFee.execute(client);

//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the new contract ID
const newContractId = receipt.contractId;

console.log("The new contract ID is " +newContractId);

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewContractCreateTransaction().
    SetGas(1000000).
    SetBytecodeFileID(byteCodeFileID)

//Sign the transaction with the client operator key and submit to a Hedera
network
txResponse, err := transaction.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the topic ID
newContractId := receipt.ContractID

fmt.Printf("The new topic ID is %v\n", newContractId)
//v2.0.0

```

```
{% endtab %}
{% endtabs %}
```

Get transaction values

Method	Type	Requirement
-----	-----	-----
getAdminKey(<keys>)	Key	Optional
getGas(<gas>)	long	Optional
getInitialBalance(<initialBalance>)	Hbar	Optional
getBytecodeFileId(<fileId>)	FileId	Optional
getProxyAccountId(<accountId>)	AccountId	Optional
getConstructorParameters(<constructorParameters>)	ByteString	Optional
getContractMemo(<memo>)	String	Optional
getDeclineStakingReward()	boolean	Optional
getStakedNodeId()	long	Optional
getStakedAccountId()	AccountId	Optional
getAutoRenewAccountId()	AccountId	Required
getAutoRenewPeriod()	Duration	Required

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
ContractCreateTransaction transaction = new ContractCreateTransaction()
    .setGas(500)
    .setBytecodeFileId(bytecodeFileId)
    .setAdminKey(adminKey);
```

```
//Get admin key
transaction.getAdminKey()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new ContractCreateTransaction()
    .setGas(500)
    .setBytecodeFileId(bytecodeFileId)
    .setAdminKey(adminKey);
```

```
//Get admin key
transaction.getAdminKey()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewContractCreateTransaction().
    SetGas(500).
    SetBytecodeFileID(byteCodeFileID).
    SetAdminKey(adminKey)
```

```
//Get admin key
transaction.GetAdminKey()
```

//v2.0.0

{% endtab %}
{% endtabs %}

delegate-contract-id.md:

Delegate Contract ID

A smart contract that, if the recipient of the active message frame, should be treated as having signed. Note that this does not mean the code being executed in the frame will belong to the given contract since it could be running another contract's code via delegatecall. Setting this key is a more permissive version of the contractID key, which also requires the code in the active message frame to belong to the contract with the given id. See HIP-206: Hedera Token Service Precompiled Contract for Hedera SmartContract Service for more details.

The delegate contract ID can be set as Key.

Constructor

Constructor	Type	Description
-----	-----	-----

new DelegateContractId(<shard, realm, num>)	long, long, long	Constructs a DelegateContractId with 0 for shardNum and realmNum (e.g., 0.0.<Num>)

Methods

Methods	Type	Description
-----	-----	-----

DelegateContractId.fromString(<id>)	String	Constructs a delegate contract ID from a String
DelegateContractId.fromBytes(<bytes>)	bytes	Constructs a delegate contract ID from bytes
DelegateContractId.fromSolidityAddress(address)	address	Constructs a delegate contract ID from Solidity address

delete-a-smart-contract.md:

Delete a smart contract

A transaction that deletes a smart contract from a Hedera network. Once a smart contract is marked deleted, you will not be able to modify any of the contract's properties. \\\ If a smart contract did not have an admin key defined, you cannot delete the smart contract. You can verify the smart contract was deleted by submitting a smart contract info query to the network. If a smart contract has an associated hbar balance, you will need to transfer the balance to another Hedera account.

Transaction Signing Requirements

If the admin key was defined for the smart contract it is required to sign the transaction.

The client operator's (fee payer account) private key is required to sign the transaction.

Transaction Fees

Please see the transaction and query NextCall a smart contract function table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	
Description		
Requirement		

setContractId(<contractId>)	ContractId	Sets the contract ID (x.z.y) which should be deleted.
	Required	
setTransferAccountId(<transferAccountId>)	AccountId	Sets the account ID (x.z.y) which will receive all remaining hbars
	Optional	
setTransferContractId(<contractId>)	ContractId	Sets the contract ID (x.z.y) which will receive all remaining hbars.
	Optional	

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
ContractDeleteTransaction transaction = new ContractDeleteTransaction()
    .setContractId(contractId);

//Freeze the transaction for signing, sign with the admin key on the contract,
sign with the client operator private key and submit to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = await new ContractDeleteTransaction()
    .setContractId(contractId)
    .freezeWith(client);

//Sign with the admin key on the contract
const signTx = await transaction.sign(adminKey)

//Sign the transaction with the client operator's private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;
```

```

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
java
//Create and freeze the transaction
transaction := hedera.NewContractDeleteTransaction().
    SetContractID(contractID)
    FreezeWith(client)

//Sign with the admin key on the contract, sign with the client operator private
key and submit to a Hedera network
txResponse. err := transaction.Sign(adminKey).Execute(client)
if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status %v\n", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

```

errors.md:

Network Response Messages

Network response messages and their descriptions.

Network Response	Description
-----	-----
CONTRACTBYTECODEEMPTY	Bytecode for smart contract is of length zero
CONTRACTDELETED	Contract is marked as deleted
CONTRACTEXECUTIONEXCEPTION	For any contract execution-related error not handled by specific error codes listed above.
CONTRACTFILEEMPTY	File to create a smart contract was of length zero
CONTRACTNEGATIVEGAS	Negative gas was offered in the smart contract call
CONTRACTNEGATIVEVALUE	Negative value / initial balance was

specified in a smart contract call / create

CONTRACTREVERTEXECUTED	Contract REVERT OP CODE executed
CONTRACTSIZELIMITEXCEEDED	Contract byte code size is over the limit
CONTRACTUPDATEFAILED	Update of the contract failed
ERRORDECODINGBYTESTRING	Decoding the smart contract binary to a byte array failed. Check that the input is a valid hex string.
EXPIRATIONREDUCTIONNOTALLOWED	The expiration date/time on a smart contract may not be reduced
INSUFFICIENTGAS	Not enough gas was supplied to execute the transaction
INSUFFICIENTLOCALCALLGAS	Payment tendered for contract local call cannot cover both the fee and the gas
INVALIDCONTRACTID	The contract id is invalid or does not exist
INVALIDPAYERACCOUNTID	The response code when a smart contract id is passed for a crypto API request
INVALIDSOLIDITYID	The solidity id is invalid or an entity with this solidity id does not exist
OBTAINERDOESNOTEXIST	TransferAccountId or transferContractId specified for contract delete does not exist
OBTAINERREQUIRED	When deleting smart contract that has crypto balance either transfer account or transfer smart contract is required
OBTAINERSAMECONTRACTID	When deleting smart contract that has crypto balance you can not use the same contract id as transferContractId as the one being deleted
LOCALCALLMODIFICATIONEXCEPTION	Local execution (query) is requested for a function that changes state
MAXCONTRACTSTORAGEEXCEEDED	Contract permanent storage exceeded the currently allowable limit
MODIFYINGIMMUTABLECONTRACT	Attempting to modify (update or delete an immutable smart contract, i.e. one created without an admin key)
NULLSOLIDITYADDRESS	Null solidity address
RESULTSIZELIMITEXCEEDED	Smart contract result size greater than specified maxResultSize

ethereum-transaction.md:

Ethereum transaction

{% hint style="info" %}

This feature is on previewnet only. API is subject to change.

{% endhint %}

The raw Ethereum transaction (RLP encoded type 0, 1, and 2) will hold signed Ethereum transactions and execute them as Hedera transactions in a prescribed manner.


```

| setEthereumData(<ethereumData>) | byte \[] |
| setMaxGasAllowanceHbar(<maxGasAllowanceHbar>) | Hbar |

{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
EthereumTransaction transaction = new EthereumTransaction()
    .setEthereumData(ethereumData)
    .setMaxGasAllowanceHbar(allowance);

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.14

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new EthereumTransaction()
    .setEthereumData(ethereumData)
    .setMaxGasAllowance(allowance);

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
const txResponse = await transaction.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status is " +transactionStatus);

//v2.14

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction, err := hedera.NewEthereumTransaction().
    SetEthereumData(ethereumData)
    SetGasAllowed(allowance)

//Sign with the client operator private key to pay for the transaction and
submit the query to a Hedera network
txResponse, err := transaction.Execute(client)
if err != nil {
    panic(err)
}

```

```
//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {the
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status %v\n", transactionStatus)

//v2.14

{% endtab %}
{% endtabs %}
```

get-a-smart-contract-function.md:

Get a smart contract function

A query that calls a function of the given smart contract instance, giving it function parameters as its inputs. This is performed locally on the particular node that the client is communicating with. It cannot change the state of the contract instance (and so, cannot spend anything from the instance's cryptocurrency account). It will not have a consensus timestamp. It cannot generate a record or a receipt. The response will contain the output returned by the function call. This is useful for calling getter functions, which purely read the state and don't change it. It is faster and cheaper than a normal call because it is purely local to a single node.

Unlike a contract execution transaction, the node will consume the entire amount of provided gas in determining the fee for this query.

Field	Description
Contract ID	The smart contract instance whose function was called.
Contract Call Result	The result returned by the function.
Error Message	The message in case there was an error during smart contract execution.
Bloom	The bloom filter for record.
Gas Used	The amount of gas used to execute the contract.
Log Info	The log info for events returned by the function.
EVM Address	The new contract's 20-byte EVM address. Only populated after release 0.23, where each created contract will have its own record. (This is an important point--the field is not repeated because there will be a separate child record for each created contract).
Gas	The amount of gas available for the call, aka the gasLimit.
Amount	Number of tinybars sent (the function must be payable if this is nonzero).
Function Parameters	The function to call and the parameters to pass to the function.
Sender Account ID	The account that is the "sender." If not present, it is the <code>accountId</code> from the transactionId.
Signer Nonce	The signer nonce field specifies what the value of the signer account nonce is post transaction execution. This value can be null.

Query Signing Requirements

The client operator account's private key (fee payer) is required to sign this

query

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Methods

Method	Type	Description	Requirement
<code>setContractId(&#x3C;contractId>)</code>		ContractId	Sets the contract instance to call, in the format used in transactions (x.z.y).
<code>setGas(&#x3C;gas>)</code>	int64		Sets the amount of gas to use for the call.
<code>setFunction(&#x3C;name>)</code>	String		Sets the function name to call. The function will be called with no parameters.
<code>setFunction(&#x3C;name, params>)</code>	String, bytes		Sets the function to call, and the parameters to pass to the function.
<code>setSenderAccountId(&#x3C;accountId>)</code>	Optional	AccountId	The account that is the "sender." If not present it is the accountId from the transactionId. Typically a different value than specified in the transactionId requires a valid signature over either the Hedera transaction or foreign transaction data.

```
{% tabs %}  
{% tab title="Java" %}
```

```
java
```

```
//Contract call query
```

```
ContractCallQuery query = new ContractCallQuery()  
    .setContractId(contractId)  
    .setGas(600)  
    .setFunction("greet");
```

```
//Sign with the client operator private key to pay for the query and submit the  
query to a Hedera network
```

```
ContractFunctionResult contractCallResult = query.execute(client);
```

```
// Get the function value
```

```
String message = contractCallResult.getString(0);  
System.out.println("contract message: " + message);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Contract call query
```

```
const query = new ContractCallQuery()  
    .setContractId(contractId)  
    .setGas(600)  
    .setFunction("greet");
```

```
//Sign with the client operator private key to pay for the query and submit the  
query to a Hedera network
```

```
const contractCallResult = await query.execute(client);
```

```
// Get the function value
```


Please use the Hedera fee estimator to estimate your query fee cost

Methods

Method	Type	Description
Requirements		
-----	-----	
-----	-----	
setContractId(<contractId>)	ContractId	The ID of the contract to return
the bytecode for Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
ContractByteCodeQuery query = new ContractByteCodeQuery()
    .setContractId(contractId);

//Sign with the client operator private key and submit to a Hedera network
ByteString bytecode = query.execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new ContractByteCodeQuery()
    .setContractId(contractId);

//Sign with the client operator private key and submit to a Hedera network
const bytecode = await query.execute(client);

{% endtab %}

{% tab title="Go" %}
java
//Create the query
query := hedera.NewContractByteCodeQuery().
    SetContractID(contractId)

//Sign with the client operator private key to pay for the query and submit the
query to a Hedera network
bytecode, err := query.Execute(client)

if err != nil {
    panic(err)
}

{% endtab %}
{% endtabs %}
```

Get query values

Method	Type	Description
Requirements		
-----	-----	
-----	-----	
getContractId(<contractId>)	ContractId	Get the contract ID on the
transaction Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
```

```
ContractByteCodeQuery query = new ContractByteCodeQuery()
    .setContractId(newContractId);
```

```
//Get the contract ID
query.getContractId()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
java
//Create the query
const query = new ContractByteCodeQuery()
    .setContractId(newContractId);
```

```
//Get the contract ID
query.getContractId()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
java
//Create the query
query := hedera.NewContractByteCodeQuery().
    SetContractID(contractId)
```

```
query.GetContractID()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
# get-smart-contract-info.md:
```

```
Get smart contract info
```

A query that returns the current state of a smart contract instance, including its balance. Queries do not change the state of the smart contract or require network consensus. The information is returned from a single node processing the query.

In Services release 0.50, Returning token balance information from the consensus node was deprecated with HIP-367. This query now returns token information by requesting the information from the Hedera Mirror Node APIs via `/api/v1/accounts/{id}/tokens`. Token symbol is not returned in the response.

```
Smart Contract Info Response
```

```
| Field | Description
```

```
|
```

```
| ----- |
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
----- |
```

```
| Contract ID | ID of the contract instance, in the format used in transactions.
```

Account ID	ID of the cryptocurrency account owned by the contract instance.
Contract Account ID	ID of both the contract instance and the cryptocurrency account owned by the contract.
Admin Key	The state of the instance and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance.
Expiration Time	The current time at which this contract instance (and its account) is set to expire.
Auto Renew Period	The expiration time will extend every this many seconds. If there are insufficient funds, then it extends as long as possible. If the account is empty when it expires, then it is deleted.
Storage	Number of bytes of storage being used by this instance (which affects the cost to extend the expiration time).
Contract Memo	The memo associated with the contract (max 100 bytes).
Balance	The current balance of the contract.
Deleted	Whether the contract has been deleted.
Ledger ID	The ID of the network the response came from. See HIP-198.
Staking Info	<p>The staking metadata for this contract. This includes the staking period start, the pending reward, the account ID or the node ID, whether or not rewards were declined, and how many hbars are staked to this contract account, if any. See HIP-406. Live on: <code>previewnet/testnet</code></p>
Token Relationships	The metadata of the tokens associated to the contract.

Query Signing Requirements

The client operator account's private key (fee payer) is required to sign this query

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee

Methods

```
<table><thead><tr><th
width="346.3333333333333">Method</th><th>Type</th><th>Description</th></tr></thead><tbody><tr><td><code>setContractId(&#x3C;contractId>)</code></td><td>ContractId</td><td>The ID of the smart contract to return the token
for</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
```

[illegible]

hedera-service-solidity-libraries.md:

Hedera Service Solidity Libraries

Hedera Token Service

Hedera Token Service integration allows you to write token transactions natively in Solidity smart contracts. There are a few Solidity source files available to developers.

```
HederaTokenService.sol
HederaResponseCodes.sol
IHederaTokenService.sol
ExpiryHelper.sol
FeeHelper.sol
KeyHelper.sol
```

The Hedera Token Service Solidity file provides the transactions to interact with tokens created on Hedera. The Hedera Response Codes contract provides the response codes associated with network errors. The IHedera Token Service is a supporting library for the Hedera Token Service Solidity file. You can grab these libraries here to add to your project and import them to your Solidity contract. Please see the example file below.

```
{% code title="ContractExample.sol" %}
solidity
// SPDX-License-Identifier: GPL-3.0
```

```
pragma solidity >=0.7.0 <0.9.0;
```

```
import "./HederaTokenService.sol";
import "./HederaResponseCodes.sol";
```

```
contract contractExample is HederaTokenService {
    ...
    int response = HederaTokenService.transferToken(tokenAddress, msg.sender,
    address(this), amount);
    ...
}
```

```
{% endcode %}
```

```
{% hint style="info" %}
```

Note: Although the IHederaTokenService.sol file is not imported in the contract, you will need it in your project directory for the supporting libraries to reference.

```
{% endhint %}
```

HederaTokenService.sol API Docs

Create Tokens

```
{% hint style="info" %}
```

HIP-358: Token create precompile is live on previewnet and testnet. The TokenCreateContract example contains four examples of how to create a token using the token create solidity libraries.

```
{% endhint %}
```

```
<mark style="color:purple;">createFungibleToken(token, initialTotalSupply,
decimals)</mark>
```

A transaction that creates a fungible token. Returns the new token address.

Param	Type	Description
token	IHederaTokenService.HederaToken memory	The basic properties of the token being created. This includes the token name, symbol, treasury account, keys, expiry, etc.
initialTotalSupply	uint	Specifies the initial supply of tokens to be put in circulation. The initial supply is sent to the Treasury Account. The supply is in the lowest denomination possible.
decimals	uint	The number of decimal places a token is divisible by.

```
<mark style="color:purple;">createFungibleTokenWithCustomFees(token,
initialTotalSupply, decimals, fixedFees, fractionalFees)</mark>
```

A transaction that creates a fungible token with custom fees. Returns the new token address.

Param	Type	Description
token	IHederaTokenService.HederaToken memory	The basic properties of the token being created. This includes the token name, symbol, treasury account, keys, expiry, etc.
initialTotalSupply	uint	Specifies the initial supply of tokens to be put in circulation. The initial supply is sent to the Treasury Account. The supply is in the lowest denomination possible.
decimals	uint	The number of decimal places a token is divisible by.
fixedFees	IHederaTokenService.FixedFee\[]	List of fixed fees to apply to the token.
fractionalFees	IHederaTokenService.FractionalFee\[]	List of fractional fees to apply to the token.

```
<mark style="color:purple;">createNonFungibleToken(token)</mark>
```

Creates a non fungible token. Returns the new token address.

Param	Type	Description
token	IHederaTokenService.HederaToken memory	The basic properties of the token being created. This includes the token name, symbol, treasury account, keys, expiry, etc.

```
<mark style="color:purple;">createNonFungibleTokenWithCustomFees(token,
fixedFees, fractionalFees)</mark>
```

Creates a non fungible token with custom fees. Returns the new token address.

Param	Type	Description
token	IHederaTokenService.HederaToken memory	The basic properties of the token being created. This includes the token name, symbol, treasury account, keys, expiry, etc.
fixedFees	IHederaTokenService.FixedFee\[]	List of fixed fees to apply to the token.
fractionalFees	IHederaTokenService.FractionalFee\[]	List of fractional fees to apply to the token.

Transfer Tokens

`<mark style="color:purple;">cryptoTransfer(tokenTransfers)</mark>`

A transaction that transfers the provided list of tokens.

ABI Version: 2

Param	Type	Description
tokenTransfers	[IHederaTokenService.TokenTransferList\[]]	(https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/system-contracts/hedera-token-service/IHederaTokenService.sol)memory The list of token transfers

`<mark style="color:purple;">transferToken(token, sender, receiver, amount)</mark>`

Transfers tokens where the calling account/contract is implicitly the first entry in the token transfer list, where the amount is the value needed to zero balance the transfers. The account address sending the token is required to sign the transaction.

ABI Version: 1

Param	Type	Description
token	address	The token ID to transfer to/from. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
sender	address	The address sending the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
receiver	address	The address of the receiver of the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
amount	int64	Non-negative value of the token to send. A negative value will result in a failure.

`<mark style="color:purple;">transferTokens(token, accountIds, amount)</mark>`

Initiates a fungible token transfer. This transaction accepts zero unit token

transfer operations for fungible tokens (HIP-564). Not applicable to non-fungible tokens.

ABI Version: 1

Param	Type	Description
token	address	The token ID to transfer. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
accountIds	address\[] memory	Hedera accounts to do a transfer to/from. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
amounts	int64\[] memory	Non-negative value to send. A negative value will result in a failure.

<mark style="color:purple;">transferNFT(token, sender, receiver, serialNum)</mark>

Transfers tokens where the calling account/contract is implicitly the first entry in the token transfer list, where the amount is the value needed to zero balance the transfers. The address sending the token is required to sign the transaction.

ABI Version: 1

Param	Type	Description
token	address	The token to transfer to/from.
sender	address	The address sending the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
receiver	address	The address of the receiver of the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
serialNum	int64	The serial number of the NFT to transfer.

<mark style="color:purple;">transferNFTs(token, sender, receiver, serialNumber)</mark>

Initiates a non-fungible token transfer

ABI Version: 1

Param	Type	Description

Param	Type	Description
token	address	The ID of the token to transfer. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
sender	address[] memory	The address sending the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
receiver	address[] memory	The address of the receiver of the token. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
serialNumber	int64	The serial number of the NFT is sent by the same index of the sender.

Mint Tokens

```
<mark style="color:purple;">mintToken(token, amount, metadata)</mark>
```

Mints an amount of the token to the defined treasury account. This transaction accepts zero-amount token mint operations for fungible tokens (HIP-564).

ABI Version: 2

Param	Type	Description
token	address	The Hedera token to mint. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
amount	uint64	Applicable to FUNGIBLE TOKENS ONLY. The amount to mint to the Treasury Account. Amount must be a positive non-zero number represented in the lowest denomination of the token. The new supply must be lower than 2^{63} .
metadata	bytes[] memory	Applicable to NON-FUNGIBLE TOKENS ONLY. Maximum allowed size of each metadata is 100 bytes

Burn Tokens

```
<mark style="color:purple;">burnToken(token, amount, serialNumbers)</mark>
```

Burns an amount of the token from the defined treasury account. This transaction accepts zero amount token burn operations for fungible tokens (HIP-564).

ABI Version: 2

Param	Type	Description
token	address	The token to burn. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.

Param	Type	Description
amount	uint64	Applicable to FUNGIBLE TOKENS ONLY. The amount to burn from the Treasury Account. Amount must be a positive non-zero number, not bigger than the token balance of the treasury account (0; balance], represented in the lowest denomination.
serialNumbers	int64\[]	Applicable to NON-FUNGIBLE TOKENS ONLY. The list of serial numbers to be burned.

Associate Tokens

```
<mark style="color:purple;">associateToken(account, tokens)</mark>
```

Associates the provided account with the provided tokens. Must be signed by the account that is associated with the token.

ABI Version: 2

Param	Type	Description
account	address	The account to be associated with the provided tokens. The address is a mapping of shard.realm.number (0.0.x) into a 20 byte Solidity address.
token	address\[]	The list of tokens to be associated with the provided account. In the case of non-fungible tokens, once an account is associated, it can hold any number of NFTs (serial numbers) of that token type. The address is a mapping of <code>shard.realm.number</code> (0.0.x) into a 20 byte Solidity address.

Dissociate Tokens

```
<mark style="color:purple;">dissociateToken(account, token)</mark>
```

Dissociates the provided account with the provided token. Must be signed by the provided Account's key.

ABI Version: 2

Param	Type	Description
account	address	The Hedera account to be dissociated from the provided token. The address is a mapping of <code>shard.realm.number</code> (0.0.x) into a 20 byte Solidity address.
token	address	The token to be dissociated from the provided account. The address is a mapping of <code>shard.realm.number</code> (0.0.x) into a 20 byte Solidity address.

```
<mark style="color:purple;">dissociateTokens(account, tokens)</mark>
```

ABI Version: 2

```
<mark style="color:purple;">allowance(token, owner, spender)</mark>
```

Param	Type	Description
token	address	The Hedera token ID in Solidity format to check the allowance of.
owner	address	The owner account in Solidity format of the tokens to be spent.
spender	address	The spender account in Solidity format.

Param	Type	Description
token	address	The Hedera token ID to approve in Solidity format.
spender	address	The spender account ID authorized to spend in Solidity format.
amount	uint256	The amount of tokens the spender is authorized to spend.

Param	Type	Description
-----	-----	-----
-----	-----	-----

Param	Type	Description
token	address	The Hedera NFT token ID in solidity format to approve.
approved	address	The Hedera account ID to approve in Solidity format. To revoke approvals pass in the zero address.
serialNumber	uint256	The NFT serial number to approve.

`<mark style="color:purple;">getApproved(token, serialNumber)</mark>`

Get the approved Hedera accounts for a single NFT. Applicable to non-fungible tokens (NFTs).

Param	Type	Description
token	address	The Hedera token ID for the NFT in Solidity format.
serialNumber	int64	The NFT serial number.

`<mark style="color:purple;">isApprovedForAll(token, owner, operator)</mark>`

Returns whether or not the operator account is approved to spend on behalf of the owner.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
owner	address	The Hedera account ID in Solidity format
operator	address	The Hedera account ID in Solidity format

`<mark style="color:purple;">setApprovalForAll(token, operator, approved)</mark>`

Enable or disable approval for a third party ("operator") to manage all of msg.sender's assets.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
operator	address	The Hedera account ID in Solidity format.
approved	bool	The Boolean data type in true or false.

`<mark style="color:purple;">isFrozen(token, account)</mark>`

Returns whether or not the account was frozen. The response will return true is the account is frozen. This means the token cannot be transferred from the account until the account is unfrozen.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

`<mark style="color:purple;">isKyc(token, account)</mark>`

Returns whether or not the token has been granted KYC.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

`<mark style="color:purple;">isToken(token)</mark>`

Returns whether or not the token exists on Hedera.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">deleteToken(token)</mark>`

Deletes the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">getTokenCustomFees(token)</mark>`

Returns the custom fees for the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">getTokenDefaultFreezeStatus(token)</mark>`

Returns the token freeze status on the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">getTokenDefaultKycStatus(token)</mark>`

Returns the KYC status on the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">getTokenExpiryInfo(token)</mark>`

Returns the token expiration for the specified token.

Param	Type	Description
token	address	The ID of the Hedera token in Solidity format.

`<mark style="color:purple;">getTokenInfo(token)</mark>`

Retrieves general token entity information for the specified token.

Param	Type	Description
token	address	The ID of the Hedera token in Solidity format.

`<mark style="color:purple;">getFungibleTokenInfo(token)</mark>`

Retrieves fungible specific token property information for a fungible token.

Param	Type	Description
token	address	The Hedera token ID of the fungible token in Solidity format.

<mark style="color:purple;">getNonFungibleTokenInfo(token)</mark>

Retrieves non-fungible specific token info for a given NFT.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

<mark style="color:purple;">getTokenKey(token, keyType)</mark>

Returns the token key for the specified key type. A key type can be the KYC key, pause key, freeze key, etc.

Param	Type	Description
token	address	The ID of the token in solidity format to return the key value.
keyType	uint	The keyType of the desired key value.

<mark style="color:purple;">getTokenType(token)</mark>

Returns the token type (fungible or non-fungible) for the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

<mark style="color:purple;">freezeToken(token, account)</mark>

Freezes the account from transacting the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

<mark style="color:purple;">unfreezeToken(token, account)</mark>

Unfreezes the account from transacting the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

<mark style="color:purple;">grantTokenKyc(token, account)</mark>

Grants KYC to the Hedera account for the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

<mark style="color:purple;">revokeTokenKyc(token, account)</mark>

Revokes KYC to the Hedera account for the specified token.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.

`<mark style="color:purple;">pauseToken(token)</mark>`

Prevents a token from being transacted if set.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">unpauseToken(token)</mark>`

Unpauses a token from a previously paused state.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.

`<mark style="color:purple;">wipeTokenAccount(token, account, amount)</mark>`

Wipes fungible tokens from the specified account. This transaction accepts zero amount token wipe operations for fungible tokens (HIP-564).

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.
amount	uint32	The amount of fungible tokens to wipe.

`<mark style="color:purple;">wipeTokenAccountNFT(token, token, serialNumbers)</mark>`

Wipes a non-fungible token from the specified account.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
account	address	The Hedera account ID in Solidity format.
serialNumbers	int64\[] memory	The NFT serial numbers to wipe from the account.

`<mark style="color:purple;">updateTokenInfo(token, tokenInfo)</mark>`

Updates the properties of a token including the name, symbol, treasury account, memo, etc.

Param	Type	Description
token	address	The Hedera token ID in Solidity format.
tokenInfo	IHederaTokenService.HederaToken	The token properties to update.

`<mark style="color:purple;">updateTokenExpiryInfo(token, expiryInfo)</mark>`

Update the token expiration time.

Param	Type
-------	------

Description	
token	address
The Hedera token ID in Solidity format.	
expiryInfo	IHederaTokenService.Expiry memory
The expiry properties of a token.	

`<mark style="color:purple;">updateTokenKeys(token, keys)</mark>`

Update the keys set on a token. The key type is defined in the key parameter.

Param	Type
Description	
token	address
The Hedera token ID in Solidity format.	
keys	[IHederaTokenService.TokenKey\[\]](https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/system-contracts/hedera-token-service/IHederaTokenService.sol) memory
The token key type.	

`<mark style="color:purple;">updateTokenKeys(token, keys)</mark>`

Update the keys set on a token. The key type is defined in the key parameter.

Param	Type
Description	
token	address
The Hedera token ID in Solidity format.	
keys	[IHederaTokenService.TokenKey\[\]](https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/system-contracts/hedera-token-service/IHederaTokenService.sol) memory
The token key type.	

Gas Cost

```
{% content-ref url="../../../networks/mainnet/fees/" %}
fees
{% endcontent-ref %}
```

Examples

```
{% content-ref url="../../../tutorials/smart-contracts/deploy-a-contract-using-the-hedera-token-service.md" %}
deploy-a-contract-using-the-hedera-token-service.md
{% endcontent-ref %}
```

README.md:

Smart Contract Service

update-a-smart-contract.md:

Update a smart contract

A transaction that allows you to modify the smart contract entity state like admin keys, proxy account, auto-renew period, and memo. This transaction does not update the contract that is tied to the smart contract entity. The contract tied to the entity is immutable. The contract entity is immutable if an admin key is not specified. Once the transaction has been successfully executed on a Hedera network the previous field values will be updated with the new ones. To get a previous state of a smart contract instance, you can query a mirror node for that data. Any null field is ignored (left unchanged)

Transaction Signing Requirements

If only the expiration time is being modified, then no signature is needed on this transaction other than for the account paying for the transaction itself. If any other smart contract entity property is being modified, the transaction must be signed by the admin key. If the admin key is being updated, the new key must sign

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Smart Contract Properties

Field	Description
Admin Key	Sets the new admin key and its fields can be modified arbitrarily if this key signs a transaction to modify it. If this is null, then such modifications are not possible, and there is no administrator that can override the normal operation of this smart contract instance. Note that if it is created with no admin keys, then there is no administrator to authorize changing the admin keys, so there can never be any admin keys for that instance. The bytecode will also be immutable.
Expiration Time	The new expiry of the contract, no earlier than the current expiry (resolves to EXPIRATIONREDUCTIONNOTALLOWED otherwise).
Staked ID	The new node account ID or node ID this contract is being staked to. If set to the sentinel 1, this removes the contract's staked node ID. See HIP-406 .
Note	Accounts cannot stake to contract accounts. This will fixed in a future release.
Decline Rewards	Updates the contract to decline receiving staking rewards if set to true. The default value is false. See HIP-406 .
Auto Renew Period	The new period that the instance will charge its account every this many seconds to renew.
Auto Renew Account ID	Indicates the account that will pay the contract's auto-renew fee. If the Auto Renew account id is cleared, then the smart contract's account will be charged the auto-renew fee.
Automatic Token Associations	The maximum number of tokens that this contract can be automatically associated with (i.e., receive air-drops from).
Memo	The new memo to be associated with this contract.

Methods

Method	Type	Requirement
<code>setContractId(contractId)</code>		HIP-406

types.md#contractid">ContractId</td><td>Required</td></tr><tr><td><code>setAdminKey(<keys>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setContractMemo(<memo>)</code></td><td>String</td><td>Optional</td></tr><tr><td><code>setExpirationTime(<expirationTime>)</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>setMaxAutomaticTokenAssociations()</code></td><td>int</td><td>Optional</td></tr><tr><td><code>setContractMemo(<memo>)</code></td><td>String</td><td>Optional</td></tr><tr><td><code>setStakedAccountId(<stakedAccountId>)</code></td><td>AccountId</td><td>Optional</td></tr><tr><td><code>setStakedNodeId(<stakedNodeId>)</code></td><td>long</td><td>Optional</td></tr><tr><td><code>setDeclineStakingReward(<declineStakingReward>)</code></td><td>boolean</td><td>Optional</td></tr><tr><td><code>setAutoRenewPeriod(<autoRenewPeriod>)</code></td><td>Duration</td><td>Optional</td></tr><tr><td><code>setAutoRenewAccountId(<accountId>)</code></td><td>AccountId</td><td>Optional</td></tr><tr><td><code>clearAutoRenewAccountId()</code></td><td></td><td>Optional</td></tr></tbody></table>

```
{% hint style="info" %}
```

Note: The new expiration time must be an instance of type Timestamp, thus, the Timestamp object has to be imported from the SDK package. The new expiration time has to be initialized as a new instance of that type.

```
{% endhint %}
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction
```

```
ContractUpdateTransaction transaction = new ContractUpdateTransaction()
    .setContractId(contractId)
    .setAdminKey(adminKey);
```

```
//Modify the default transaction fee
```

```
ContractUpdateTransaction modifyTransactionFee =
transaction.setMaxTransactionFee(new Hbar(20));
```

```
//sign with the new admin key, sign with the old admin key, sign with the client
operator account and submit to a Hedera network
```

```
TransactionResponse txResponse =
modifyTransactionFee.freezeWith(client).sign(newAdminKey).sign(adminKey).execute
(client);
```

```
//Get the consensus status of the transaction
```

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The consensus status of the transaction is "
+transactionStatus);
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Create the transaction
```

```
const transaction = await new ContractUpdateTransaction()
    .setContractId(contractId)
    .setAdminKey(adminKey)
    .setMaxTransactionFee(new Hbar(20))
    .freezeWith(client);
```

```

//Sign the transaction with the old admin key and new admin key
const signTx = await (await transaction.sign(newAdminKey)).sign(adminKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client)

//Get the consensus status of the transaction
const transactionStatus = receipt.status;

console.log("The consensus status of the transaction is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewContractUpdateTransaction().
    SetContractID(contractId).
    SetBytecodeFileID(byteCodeFileID).
    SetAdminKey(adminKey)

//Change the default max transaction fee to 20 HBAR
modifyMaxTransactionFee := transaction.SetMaxTransactionFee(hedera.HbarFrom(20,
hedera.HbarUnits.Hbar))

//Freeze the transaction
freezeTransaction, err := modifyMaxTransactionFee.FreezeWith(client)
if err != nil {
    panic(err)
}

//Sign with the old admin key, sign with the new admin key, sign with the client
operator account and submit to a Hedera network
txResponse, err :=
freezeTransaction.Sign(newAdminKey).Sign(adminKey).Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

fmt.Printf("The transaction consensus status %v\n", transactionStatus)

//v2.0.0

{% endtab %}
{% endtabs %}

Get transaction values

```

Method	Type	Requirement
<code>getContractId()</code>	ContractId	Required
<code>getAdminKey()</code>	Key	Optional
<code>getBytecodeFileId()</code>	FileId	Optional
<code>getProxyAccountId()</code>	AccountId	Optional
<code>getContractMemo()</code>	String	Optional
<code>getStakedAccountId()</code>	AccountId	Optional
<code>getStakedNodeId()</code>	long	Optional
<code>getDeclineStakingReward()</code>	boolean	Optional
<code>getAutoRenewAccountId()</code>	AccountId	Required
<code>getAutoRenewPeriod()</code>	Duration	Required

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction with an admin key
ContractUpdateTransaction transaction = new ContractUpdateTransaction()
    .setContractId(contractId)
    .setAdminKey(adminKey);
```

```
//Get admin key
transaction.getAdminKey()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
java
//Create the transaction with an admin key
const transaction = new ContractUpdateTransaction()
    .setContractId(contractId)
    .setAdminKey(adminKey);
```

```
//Get admin key
transaction.getAdminKey()
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewContractUpdateTransaction().
    SetContractID(contractId).
    SetBytecodeFileID(byteCodeFileID).
    SetAdminKey(adminKey)
```

```
//Get admin key
transaction.GetAdminKey()
```

```
//v2.0.0
```

```
{% endtab %}
{% endtabs %}
```

associate-tokens-to-an-account.md:

Associate tokens to an account

Associates the provided Hedera account with the provided Hedera token(s). Hedera accounts must be associated with a fungible or non-fungible token first before you can transfer tokens to that account. When you transfer a custom fungible or non-fungible token to the alias account ID, the token association step is skipped and the account will automatically be associated with the token upon creation. In the case of NON\FUNGIBLE Type, once an account is associated, it can hold any number of NFTs (serial numbers) of that token type. The Hedera account that is associated with a token is required to sign the transaction.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALIDTOKENREF.

If any of the provided tokens has been deleted, the transaction will resolve to TOKENWASDELETED.

If an association between the provided account and any of the tokens already exists, the transaction will resolve to TOKENALREADYASSOCIATEDTOACCOUNT.

If the provided account's associations count exceeds the constraint of maximum token associations per account, the transaction will resolve to TOKENSPERACCOUNTLIMITEXCEEDED.

On success, associations between the provided account and tokens are made and the account is ready to interact with the tokens.

{% hint style="info" %}

There is currently no limit on the number of token IDs that can be associated with an account (reference HIP-367). Still, you can see TOKENSPERACCOUNTLIMITEXCEEDED responses for pre-HIP-367 transactions.

{% endhint %}

Transaction Signing Requirements

The key of the account the token is being associated to
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	-----
setAccountId(<accountId>)	AccountId	The account to be associated with the provided tokens
Required		
setTokenIds(<tokens>)	List \<TokenId>	The tokens to be associated with the provided account
Required		

{% tabs %}

{% tab title="Java" %}

java

//Associate a token to an account

```
TokenAssociateTransaction transaction = new TokenAssociateTransaction()
    .setAccountId(accountId)
    .setTokenIds(Collections.singletonList(tokenId));
```

```

//Freeze the unsigned transaction, sign with the private key of the account that
is being associated to a token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(accountKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status " +transactionStatus);
//v2.0.4

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Associate a token to an account and freeze the unsigned transaction for
signing
const transaction = await new TokenAssociateTransaction()
    .setAccountId(accountId)
    .setTokenIds([tokenId])
    .freezeWith(client);

//Sign with the private key of the account that is being associated to a token
const signTx = await transaction.sign(accountKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Associate the token to an account and freeze the unsigned transaction for
signing
transaction, err := hedera.NewTokenAssociateTransaction().
    SetAccountID(accountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the private key of the account that is being associated to a token,
submit the transaction to a Hedera network
txResponse, err = transaction.Sign(accountKey).Execute(client)

if err != nil {
    panic(err)
}

```

```
//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)
```

```
if err != nil {
    panic(err)
}
```

```
//Get the transaction consensus status
status := receipt.Status
```

```
fmt.Printf("The transaction consensus status is %v\n", status)
```

```
//v2.1.0
```

```
{% endtab %}
{% endtabs %}
```

```
# atomic-swaps.md:
```

```
Atomic swaps
```

An atomic swap is when you swap tokens between two accounts without using a third-party intermediary, such as a centralized exchange or custody provider, to facilitate the transfer. Native tokens issued using the Hedera Token Service (HTS) can be swapped with another or with HBAR in a single transaction using the TransferTransaction API call. For each atomic swap within a single transaction, you'll need to designate an account to be debited (-) any number of tokens and the corresponding account which will receive those tokens.

```
Signing Requirements
```

The private keys for the accounts which are being debited tokens are required to sign the transaction.

```
{% hint style="info" %}
Hedera accounts must be associated with the specified token before you can
transfer a token to their account. Please see how to associate a token to an
account here.
{% endhint %}
```

```
{% tabs %}
{% tab title="Java" %}
java
```

```
//Atomic swap between a Hedera Token Service token and hbar
TransferTransaction atomicSwap = new TransferTransaction()
    .addHbarTransfer(accountId1, new Hbar(-1))
    .addHbarTransfer(accountId2, new Hbar(1))
    .addTokenTransfer(tokenId, accountId2, -1)
    .addTokenTransfer(tokenId, accountId1, 1)
    .freezeWith(client);
```

```
//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
TransactionResponse txResponse =
atomicSwap.sign(accountKey1).sign(accountKey2).execute(client);
```

```
//-----OR-----
```

```
//Atomic swap between two hedera Token Service created tokens
TransferTransaction atomicSwap = new TransferTransaction()
    .addTokenTransfer(tokenId1, accountId1, -1)
    .addTokenTransfer(tokenId1, accountId2, 1)
```

```

        .addTokenTransfer(tokenId2, accountId2, -1)
        .addTokenTransfer(tokenId2, accountId1, 1)
        .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
TransactionResponse txResponse =
atomicSwap.sign(accountKey1).sign(accountKey2).execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Atomic swap between a Hedera Token Service token and hbar
const atomicSwap = await new TransferTransaction()
    .addHbarTransfer(accountId1, new Hbar(-1))
    .addHbarTransfer(accountId2, new Hbar(1))
    .addTokenTransfer(tokenId, accountId2, -1)
    .addTokenTransfer(tokenId, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
const txResponse = await (await (await
atomicSwap.sign(accountKey1)).sign(accountKey2)).execute(client);

//-----OR-----

//Atomic swap between two hedera Token Service created tokens
const atomicSwap = await new TransferTransaction()
    .addTokenTransfer(tokenId1, accountId1, -1)
    .addTokenTransfer(tokenId1, accountId2, 1)
    .addTokenTransfer(tokenId2, accountId2, -1)
    .addTokenTransfer(tokenId2, accountId1, 1)
    .freezeWith(client);

//Sign the transaction with accountId1 and accountId2 private keys, submit the
transaction to a Hedera network
const txResponse = await (await (await
atomicSwap.sign(accountKey1)).sign(accountKey2)).execute(client);

{% endtab %}

{% tab title="Go" %}
go
//Atomic swap between a Hedera Token Service token and hbar
atomicSwap, err := hedera.NewTransferTransaction().
    AddHbarTransfer(accountId1, hedera.NewHbar(-1)).
    AddHbarTransfer(accountId2, hedera.NewHbar(1)).
    AddTokenTransfer(tokenId, accountId2, -1).
    AddTokenTransfer(tokenId, accountId1, 1).
    FreezeWith(client)

txResponse, err :=
atomicSwap.Sign(accountKey1).Sign(accountKey2).Execute(client)

//-----OR-----

//Atomic swap between two hedera Token Service created tokens
atomicSwap, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId1, accountId1, -1).
    AddTokenTransfer(tokenId1, accountId2, 1).
    AddTokenTransfer(tokenId2, accountId2, -1).
    AddTokenTransfer(tokenId2, accountId1, 1).

```



```
FreezeWith(client)
```

```
txResponse, err :=  
atomicSwap.Sign(accountKey1).Sign(accountKey2).Execute(client)
```

```
{% endtab %}  
{% endtabs %}
```

```
# burn-a-token.md:
```

```
Burn a token
```

Burns fungible and non-fungible tokens owned by the Treasury Account. If no Supply Key is defined, the transaction will resolve to TOKEN\HAS\NO\SUPPLY\KEY.

The operation decreases the Total Supply of the Token.

Total supply cannot go below zero.

The amount provided must be in the lowest denomination possible.

Example: Token A has 2 decimals. In order to burn 100 tokens, one must provide an amount of 10000. In order to burn 100.55 tokens, one must provide an amount of 10055.

This transaction accepts zero unit token burn operations for fungible tokens (HIP-564)

Transaction Signing Requirements

Supply key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee

Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		

-----	-----	-----
-----	-----	-----
setTokenId(<tokenId>)	TokenId	The ID of the token to burn supply
Required		
setAmount(<amount>)	long	The number of tokens to burn (fungible
tokens)	Optional	
setSerials(<serials>)	List<long>	Applicable to tokens of type
NONFUNGIBLEUNIQUE. The list of NFT serial IDs to burn.	Optional	
addSerial(<serial>)	long	Applicable to tokens of type
NONFUNGIBLEUNIQUE. The serial ID to burn.	Optional	

```
{% tabs %}  
{% tab title="Java" %}
```

```
java  
//Burn 1,000 tokens  
TokenBurnTransaction transaction = new TokenBurnTransaction()  
    .setTokenId(tokenId)  
    .setAmount(1000);
```

```
//Freeze the unsigned transaction, sign with the supply private key of the  
token, submit the transaction to a Hedera network
```

```
TransactionResponse txResponse =  
transaction.freezeWith(client).sign(supplyKey).execute(client);
```

```

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Burn 1,000 tokens and freeze the unsigned transaction for manual signing
const transaction = await new TokenBurnTransaction()
    .setTokenId(tokenId)
    .setAmount(1000)
    .freezeWith(client);

//Sign with the supply private key of the token
const signTx = await transaction.sign(supplyKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Burn 1,000 tokens and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenBurnTransaction().
    SetTokenID(tokenId).
    SetAmount(1000).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the supply private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(supplyKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

```

```
//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}
```

custom-token-fees.md:

Custom token fees

When creating a token, you can configure up to 10 custom fees, automatically disbursed to specified fee collector accounts each time the token is transferred programmatically. These fees can be fixed, fractional, or royalty-based, offering revenue generation, profit-sharing, and behavior incentivization for creators. This guide is your comprehensive resource for understanding types, implementation, and best practices for custom fees on Hedera.

Types of Custom Fees

Fixed Fee: Paid by the sender of the fungible or non-fungible tokens. A fixed fee transfers a set amount to a fee collector account each time a token is transferred, independent of the transfer size. This fee can be collected in HBAR or another Hedera token but not in NFTs.

Fractional Fee: Take a specific portion of the transferred fungible tokens, with optional minimum and maximum limits. The token receiver (fee collector account) pays these fees by default. However, if `netoftransfers` is set to true, the sender pays the fees and the receiver collects the full token transfer amount. If this field is set to false, the receiver pays for the token custom fees and gets the remaining token balance.

Royalty Fee: Paid by the receiver account that is exchanging the fungible value for the NFT. When the NFT sender does not receive any fungible value, the fallback fee is charged to the NFT receiver.

```
{% hint style="info" %}
```

Note: In addition to the custom token fee payment, the sender account must pay for the token transfer transaction fee in HBAR. The "Payment of Custom Fees & Transaction Fees in HBAR" section below covers the distinction between custom fees and transaction fees.

```
{% endhint %}
```

Implementation Methods

Fixed Fee

A fixed fee entails transferring a specified token amount to predefined fee collector accounts each time a token transfer is initiated. This fee amount doesn't depend on the volume of tokens being transferred. The creator has the flexibility to collect the fee in HBAR or another fungible Hedera token. However, it's important to note that NFTs cannot be used as a token type to collect this fee. A custom fixed fee can be set for fungible and non-fungible token types.

width="409">Constructor</th><th>Description</th></tr></thead> <tbody> <tr> <td><code>new CustomFixedFee()</code></td><td>Initializes the <code>CustomFixedFee</code> object</td></tr> </td></tr></tbody>	<code>new CustomFixedFee()</code></td><td>Initializes the <code>CustomFixedFee</code> object</td></tr>
<code>new CustomFixedFee()</code></td><td>Initializes the <code>CustomFixedFee</code> object</td></tr>	

```
java
new CustomFixedFee()
```

width="306.3333333333333">Methods</th><th width="213">Description</th><th width="118" align="center">Type</th><th align="center">Requirement</th></tr>				
<code>setFeeCollectorAccountId</code></td><td>Sets the fee collector account ID that collects the fee.</td><td align="center">AccountId</td><td align="center">Required</td></tr> <tr> <td><code>setHbarAmount</code></td><td>Set the amount of HBAR to be collected.</td><td align="center">HBAR</td><td align="center">Optional</td></tr> <tr> <td><code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr></td></tr></td></tr>	<code>setHbarAmount</code></td><td>Set the amount of HBAR to be collected.</td><td align="center">HBAR</td><td align="center">Optional</td></tr> <tr> <td><code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr></td></tr>	<code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr>	<code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr>	<code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr>
<code>setHbarAmount</code></td><td>Set the amount of HBAR to be collected.</td><td align="center">HBAR</td><td align="center">Optional</td></tr> <tr> <td><code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr></td></tr>	<code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr>	<code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr>	<code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr>	
<code>setAmount</code></td><td>Sets the amount of tokens to be collected as the fee.</td><td align="center">int64</td><td align="center">Optional</td></tr> <tr> <td><code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr></td></tr>	<code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr>	<code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr>		
<code>setDenominatingTokenId</code></td><td>The ID of the token used to charge the fee. The denomination of the fee is taken as HBAR if left unset.</td><td align="center">TokenId</td><td align="center">Optional</td></tr> <tr> <td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr> </td></tr>	<code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr>			
<code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr>				

```
{% tabs %}
{% tab title="Java" %}
java
//Create a custom token fixed fee
new CustomFixedFee()
    .setAmount(1) // 1 token is transferred to the fee collecting account each
time this token is transferred
    .setDenominatingTokenId(tokenId) // The token to charge the fee in
    .setFeeCollectorAccountId(feeCollectorAccountId); // 1 token is sent to this
account everytime it is transferred

//Version: 2.0.143

{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create a custom token fixed fee
new CustomFixedFee()
    .setAmount(1) // 1 token is transferred to the fee collecting account each
time this token is transferred
    .setDenominatingTokenId(tokenId) // The token to charge the fee in
    .setFeeCollectorAccountId(feeCollectorAccountId); // 1 token is sent to this
account everytime it is transferred

//Version: 2.0.30

{% endtab %}
```

```
{% tab title="Go" %}
go
//Create a custom token fixed fee
[]hedera.Fee{
```

```

        hedera.NewCustomFixedFee().
        SetAmount(1). // 1 token is transferred to the fee collecting
account each time this token is transferred
        SetDenominatingTokenID(tokenId). // The token to charge the fee in
        SetFeeCollectorAccountID(feeCollectorAccountId) // 1 token is sent
to this account everytime it is transferred
    },
}
//Version: 2.1.16

```

```

{% endtab %}
{% endtabs %}

```

Fractional Fee

Fractional fees involve the transfer of a specified fraction of the tokens' total value to the designated fee collector account. You can set a custom fractional fee and impose minimum and maximum fee limits per transfer transaction. The fractional fee has to be less than or equal to 1. It cannot exceed the fractional range of a 64-bit signed integer. Applicable to fungible tokens only.

width="280.33333333333326">Methods	width="222">Description	width="110" align="center">Type	align="center">Requirement
<code>setFeeCollectorAccountId</code>	Sets the fee collector account ID that collects the fee.	AccountID	Required
<code>setNumerator</code>	Sets the numerator of the fraction.	long	Required
<code>setDenominator</code>	Sets the denominator of the fraction. Cannot be zero.	long	Required
<code>setMax</code>	The maximum fee that can be charged, regardless of the fractional value.	long	Optional
<code>setMin</code>	The minimum fee that can be charged, regardless of the fractional value.	long	Optional
<code>setAssessmentMethod</code>	If true, sender pays fees and the receiver collects the full token transfer amount. If false, receiver pays fees and gets remaining token balance.	boolean	Optional
<code>FeeAssessmentMethod</code>			
<code>setAllCollectorsAreExempt</code>	If true, exempts all the token's fee collector accounts from this fee.	boolean	Optional

```

{% tabs %}
{% tab title="Java" %}
java
//Create a custom token fractional fee
new CustomFractionalFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFeeCollectorAccountID(feeCollectorAccountId); // The account collecting
the 10% custom fee each time the token is transferred

```

```

//Version: 2.0.14

```

```

{% endtab %}

```

```
{% tab title="JavaScript" %}
javascript
//Create a custom token fractional fee
new CustomFractionalFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFeeCollectorAccountId(feeCollectorAccountId); // The account collecting
the 10% custom fee each time the token is transferred
```

```
//Version: 2.0.30
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
//Create a custom token fractional fee
[]hedera.Fee{
    hedera.NewCustomFractionalFee().
        SetNumerator(1). // The numerator of the fraction
        SetDenominator(10). // The denominator of the fraction
        SetFeeCollectorAccountId(feeCollectorAccountId), // The account
collecting the 10% custom fee each time the token is transferred
}
```


```
//Version: 2.1.16
```

```
{% endtab %}
{% endtabs %}
```

Royalty Fee

The royalty fee is assessed and applied each time the ownership of an NFT is transferred and is a fraction of the value exchanged for the NFT. If no value is exchanged for the NFT, a fallback fee can be imposed on the receiving account. This fee type only applies to non-fungible tokens.

```
{% hint style="info" %}
```

 NOTE: Royalty fees are strictly a convenience feature. The network can't enforce royalties if counterparties decide to split their NFT exchange into separate transactions. The NFT sender and receiver must both sign a single CryptoTransfer to ensure the proper application of royalties. There is an open HIP discussion about broadening the class of transactions for which the network automatically collects royalties. If this topic interests or concerns you, your participation in the discussion is welcome.

```
{% endhint %}
```

width="394">Constructor	Description
<code>new CustomRoyaltyFee()</code>	Initializes the <code>CustomRoyaltyFee</code> object

```
java
new CustomRoyaltyFee()
```

width="287.33333333333326">Methods	width="223">Description	width="110" align="center">Type	align="center">Requirement
<code>setFeeCollectorAccountId()</code>	Sets the fee collector account ID that collects the fee.	AccountID	Required
<code>setNumerator()</code>	Sets the numerator of the fraction.	long	Required
<code>setDenominator()</code>			

td><td>Sets the denominator of the fraction.</td><td align="center">long</td><td align="center">Required</td></tr><tr><td><code>setFallbackFee</code></td><td>If present, the fixed fee to assess to the NFT receiver when no fungible value is exchanged with the sender</td><td align="center">FixedFee</td><td align="center">Optional</td></tr><tr><td><code>setAllCollectorsAreExempt</code></td><td>If true, exempts all the token's fee collector accounts from this fee.</td><td align="center">boolean</td><td align="center">Optional</td></tr></tbody></table>

```
{% tabs %}
{% tab title="Java" %}
java
//Create a royalty fee
new CustomRoyaltyFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1)) // The
fallback fee
    .setFeeCollectorAccountId(feeCollectorAccountId))) // The account that will
receive the royalty fee
```

```
// v2.0.14
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Create a royalty fee
new CustomRoyaltyFee()
    .setNumerator(1) // The numerator of the fraction
    .setDenominator(10) // The denominator of the fraction
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1)) // The
fallback fee
    .setFeeCollectorAccountId(feeCollectorAccountId))) // The account that will
receive the royalty fee
```

```
// v2.0.30
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
//Create a royalty fee
[]hedera.Fee{
    hedera.NewCustomRoyaltyFee().
        SetFeeCollectorAccountID(feeCollectorAccountId). // The account that
will receive the royalty fee
        SetNumerator(1). // The numerator of the fraction
        SetDenominator(10). // The denominator of the fraction
        SetFallbackFee( // The fallback fee
            hedera.NewCustomFixedFee().
                SetFeeCollectorAccountID(feeCollectorAccountId).
                SetAmount(1),
        ),
}
```

```
// v2.1.16
```

```
{% endtab %}
{% endtabs %}
```

Payment of Custom Fees vs. Transaction Fees in HBAR

Understanding the difference between custom fees and standard transaction fees in HBAR is crucial for token issuers and developers working with Hedera. Custom fees are designed to enforce complex fee structures, such as royalties and fractional ownership. These fees can be fixed, fractional, or royalty-based and are usually paid in the token being transferred, although other Hedera tokens or HBAR can also be used. You can configure up to 10 custom fees automatically disbursed to designated fee collector accounts.

On the other hand, transaction fees in HBAR serve a different purpose: they compensate network nodes for processing transactions. These fees are uniform across all transaction types and are paid exclusively in HBAR. Unlike custom fees, which can be configured by the user, transaction fees are fixed by the network.

The key differences lie in their flexibility, payee, currency, and configurability. Custom fees offer greater flexibility and can be paid to any account in various tokens, and are user-defined. Transaction fees are network-defined, less flexible, and go solely to network nodes, paid only in HBAR.

Fee Exemptions

Fee collector accounts can be exempt from paying custom fees. To enable this, you need to set the exemption during the creation of the custom fees (HIP-573). If not enabled, custom fees will only be exempt for an account if that account is set as a fee collector.

Limits and Constraints

When it comes to setting custom fees, there are a few limits and constraints to keep in mind:

- First, fees cannot be set to a negative value.
- Each token can have up to 10 different custom fees.
- Additionally, the treasury account for a given token is automatically exempt from paying these custom transaction fees.
- The system also permits, at most, two "levels" of custom fees. That means a token being transferred might require fees in another token that also has its own fee schedule; however, this can only be nested two layers deep to prevent excessive complexity.

define-a-token.md:

Create a token

```
{% hint style="info" %}
Check out the "Getting Started with the Hedera Token Service" video tutorial in
JavaScript here.
{% endhint %}
```

Create a new fungible or non-fungible token (NFT) on the Hedera network. After you submit the transaction to the Hedera network, you can obtain the new token ID by requesting the receipt.

You can also create, access, or transfer HTS tokens using smart contracts - see Hedera Service Solidity Libraries and Supported ERC Token Standards.

```
{% hint style="warning" %}
```

Token Keys

If any of the token key types (KYC key, Wipe key, Metadata key, etc) are not

set during the creation of the token, you will not be able to update the token and add them in the future

If any of the token key types (KYC key, Wipe key, Metadata key, etc) are set during the creation of the token, you will not be able to remove them in the future

{% endhint %}

NFTs

For non-fungible tokens, the token ID represents an NFT class. Once the token is created, you must mint each NFT using the token mint operation.

{% hint style="warning" %}

Note: The initial supply for an NFT is required to be set to 0.

{% endhint %}

Token Properties

Property	Description
Name	Set the publicly visible name of the token. The token name is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). The token name is not unique. Maximum of 100 characters.
Type	The type of token to create. Either fungible or non-fungible.
Symbol	The publicly visible token symbol. Set the publicly visible name of the token. The token symbol is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). The token symbol is not unique. Maximum of 100 characters.
Decimal	The number of decimal places a token is divisible by. This field can never be changed.
Initial Supply	Specifies the initial supply of fungible tokens to be put in circulation. The initial supply is sent to the Treasury Account. The maximum supply of tokens is <code>9,223,372,036,854,775,807</code> (<code>2⁶³-1</code>) tokens and is in the lowest denomination possible. For creating an NFT, you must set the initial supply to 0.
Treasury Account	The account which will act as a treasury for the token. This account will receive the specified initial supply and any additional tokens that are minted. If tokens are burned, the supply will decrease from the treasury account.
Admin Key	The key which can perform token update and token delete operations on the token. The admin key has the authority to change the supply key, freeze key, pause key, wipe key, and KYC key. It can also update the treasury account of the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted).
KYC Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
Freeze Key	The key which can sign to freeze or unfreeze an account for token transactions. If empty, freezing is not possible.
Wipe Key	The key which can wipe the token balance of an account. If empty, wipe is not possible.
Supply Key	The key which can change the total supply of a token. This key is used to authorize token mint and burn transactions. If this is left empty, minting/burning tokens is not possible.
Fee Schedule Key	The key which can change the token's custom fee schedule. It must sign a <code>TokenFeeScheduleUpdate</code> transaction. A custom fee schedule token without a fee schedule key is immutable.
Pause Key	The key which has the authority to pause or unpauses a token. Pausing a token prevents the token from participating in all transactions.
Custom Fees	Custom fees to charge

during a token transfer transaction that transfers units of this token. Custom fees can either be [fixed](custom-token-fees.md#fixed-fee), [fractional](custom-token-fees.md#fractional-fee), or [royalty](custom-token-fees.md#royalty-fee) fees. You can set up to a maximum of 10 custom fees.

Max Supply	For tokens of type <code>FUNGIBLECOMMON</code> - the maximum number of tokens that can be in circulation. For tokens of type <code>NONFUNGIBLEUNIQUE</code> - the maximum number of NFTs (serial numbers) that can be minted. This field can never be changed.
Supply Type	Specifies the token supply type. Defaults to INFINITE.
Freeze	The default Freeze status (frozen or unfrozen) of Hedera accounts relative to this token. If true, an account must be unfrozen before it can receive the token.
Expiration Time	The epoch second at which the token should expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the <code>autoRenewPeriod</code> . The default expiration time is 7,890,000 seconds (90 days).
Auto Renew Account	An account which will be automatically charged to renew the token's expiration, at <code>autoRenewPeriod</code> interval. This key is required to sign the transaction if present. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
Auto Renew Period	The interval at which the auto-renew account will be charged to extend the token's expiry. The default auto-renew period is 7,890,000 seconds. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: <code>AUTORENEWDURATIONNOTINRANGE.</code>	
Memo	A short publicly visible memo about the token.
Metadata Key	The key which can update the metadata of an NFT. This key is used to sign and authorize the transaction to update the metadata of dynamic NFTs. This value can be null.
Metadata	The metadata of the token. The admin key or metadata key can be used to update this property.

Property	Description
Name	Set the publicly visible name of the token. The token name is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). The token name is not unique. Maximum of 100 characters.
Token Type	The type of token to create. Either fungible or non-fungible.
Symbol	The publicly visible token symbol. Set the publicly visible name of the token. The token symbol is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). The token symbol is not unique. Maximum of 100 characters.
Decimal	The number of decimal places a token is divisible by. This field can never be changed.
Initial Supply	Specifies the initial supply of fungible tokens to be put in circulation. The initial supply is sent to the Treasury Account. The maximum supply of tokens is <code>9,223,372,036,854,775,807</code> (<code>2^63-1</code>) tokens and is in the lowest denomination possible. For creating an NFT, you must set the initial supply to 0.
Treasury Account	The account which will act as a treasury for the token. This account will receive the specified initial supply and any additional tokens that are minted. If tokens are burned, the supply will decreased from the treasury account.
Admin Key	The key which can perform token update and token delete operations on the token. The admin key has the authority to change the supply key, freeze key, pause key, wipe key, and KYC

key. It can also update the treasury account of the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted).	
KYC Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
Freeze Key	The key which can sign to freeze or unfreeze an account for token transactions. If empty, freezing is not possible.
Wipe Key	The key which can wipe the token balance of an account. If empty, wipe is not possible.
Supply Key	The key that can change a token's total supply is used to authorize token minting and burning transactions. If this key is left empty, minting/burning tokens is not possible. If set, it can update the metadata of NFTs held in treasury.
Fee Schedule Key	The key which can change the token's custom fee schedule. It must sign a TokenFeeScheduleUpdate transaction. A custom fee schedule token without a fee schedule key is immutable.
Pause Key	The key which has the authority to pause or unpause a token. Pausing a token prevents the token from participating in all transactions.
Custom Fees	Custom fees to charge during a token transfer transaction that transfers units of this token. Custom fees can either be fixed , fractional , or royalty fees. You can set up to a maximum of 10 custom fees.
Max Supply	For tokens of type <code>FUNGIBLECOMMON</code> - the maximum number of tokens that can be in circulation. For tokens of type <code>NONFUNGIBLEUNIQUE</code> - the maximum number of NFTs (serial numbers) that can be minted. This field can never be changed. You must set the token supply type to FINITE if you set this field.
Supply Type	Specifies the token supply type. Defaults to INFINITE.
Freeze Default	The default Freeze status (frozen or unfrozen) of Hedera accounts relative to this token. If true, an account must be unfrozen before it can receive the token.
Expiration Time	The epoch second at which the token should expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod. The default expiration time is 7,890,000 seconds (90 days).
Auto Renew Account	An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval. This key is required to sign the transaction if present. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
Auto Renew Period	The interval at which the auto-renew account will be charged to extend the token's expiry. The default auto-renew period is 7,890,000 seconds. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
NOTE: The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: AUTORENEWDURATIONNOTINRANGE.	
Memo	A short publicly visible memo about the token.
Metadata Key	The key which can update the metadata of an NFT. This key is used to sign and authorize the transaction to update the metadata of dynamic NFTs. This value can be null.
Metadata	The metadata of the token. The admin key or metadata key can be used to update this property.

Transaction Signing Requirements

Treasury key is required to sign
Admin key, if specified
Transaction fee payer key

Transaction Fees

For fungible tokens, a CryptoTransfer fee is added to transfer the newly created token to the treasury account

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

```
<table><thead><tr><th
width="447.3333333333333">Method</th><th>Type</th><th>Requirement</th></tr></thead><tbody><tr><td><code>setTokenName(&#x3C;name>)</code></td><td>String</td><td>Required</td></tr><tr><td><code>setTokenType(&#x3C;tokenType>)</code></td><td><a href="token-types.md">TokenType</a></td><td>Optional</td></tr><tr><td><code>setTokenSymbol(&#x3C;symbol>)</code></td><td>String</td><td>Required</td></tr><tr><td><code>setDecimals(&#x3C;decimal>)</code></td><td>int</td><td>Optional</td></tr><tr><td><code>setInitialSupply(&#x3C;initialSupply>)</code></td><td>int</td><td>Optional</td></tr><tr><td><code>setTreasuryAccountId(&#x3C;treasury>)</code></td><td><a href=".../deprecated/sdks/specialized-types.md#accountId">AccountId</a></td><td>Required</td></tr><tr><td><code>setAdminKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setKycKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setFreezeKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setWipeKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setSupplyKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setPauseKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setFreezeDefault(&#x3C;freeze>)</code></td><td>boolean</td><td>Optional</td></tr><tr><td><code>setExpirationTime(&#x3C;expirationTime>)</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>setFeeScheduleKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setCustomFees(&#x3C;customFees>)</code></td><td>List&#x3C;<a href="custom-token-fees.md#custom-fee">CustomFee</a></td><td>Optional</td></tr><tr><td><code>setSupplyType(&#x3C;supplyType>)</code></td><td>TokenSupplyType</td><td>Optional</td></tr><tr><td><code>setMaxSupply(&#x3C;maxSupply>)</code></td><td>long</td><td>Optional</td></tr><tr><td><code>setTokenMemo(&#x3C;memo>)</code></td><td>String</td><td>Optional</td></tr><tr><td><code>setAutoRenewAccountId(&#x3C;account>)</code></td><td><a href=".../deprecated/sdks/specialized-types.md#accountId">AccountId</a></td><td>Optional</td></tr><tr><td><code>setAutoRenewPeriod(&#x3C;period>)</code></td><td>Duration</td><td>Optional</td></tr><tr><td><code>setMetadataKey(&#x3C;key>)</code></td><td>Key</td><td>Optional</td></tr><tr><td><code>setMetadata(&#x3C;bytes>)</code></td><td>bytes</td><td>Optional</td></tr></tbody></table>
```

{% hint style="info" %}

Note: Where the Admin, Pause, Freeze, and Wipe keys are left blank, the Supply key will be required as a minimum.

{% endhint %}

{% tabs %}

{% tab title="Java" %}

java

//Create the transaction

```
TokenCreateTransaction transaction = new TokenCreateTransaction()
    .setTokenName("Your Token Name")
    .setTokenSymbol("F")
    .setTreasuryAccountId(treasuryAccountId)
```

```

        .setInitialSupply(5000)
        .setAdminKey(adminKey.getPublicKey())
        .setMetadataKey(metadataKey)
        .setMetadata(metadata)
        .setMaxTransactionFee(new Hbar(30)); //Change the default max
transaction fee

//Build the unsigned transaction, sign with admin private key of the token, sign
with the token treasury private key, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).sign(treasuryKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the token ID from the receipt
TokenId tokenId = receipt.tokenId;

System.out.println("The new token ID is " + tokenId);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze for manual signing
const transaction = await new TokenCreateTransaction()
    .setTokenName("Your Token Name")
    .setTokenSymbol("F")
    .setTreasuryAccountId(treasuryAccountId)
    .setInitialSupply(5000)
    .setAdminKey(adminPublicKey)
    .setMetadataKey(metadataKey)
    .setMetadata(metadata)
    .setMaxTransactionFee(new Hbar(30)) //Change the default max transaction
fee
    .freezeWith(client);

//Sign the transaction with the token adminKey and the token treasury account
private key
const signTx = await (await transaction.sign(adminKey)).sign(treasuryKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponse = await signTx.execute(client);

//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the token ID from the receipt
const tokenId = receipt.tokenId;

console.log("The new token ID is " + tokenId);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction
tokenCreateTransaction, err := hedera.NewTokenCreateTransaction().
    SetTokenName("Your Token Name").

```

```

        SetTokenSymbol("F").
        SetTreasuryAccountID(treasuryAccountId).
        SetInitialSupply(1000).
        SetAdminKey(adminKey).
        SetMetadataKey(metadataKey).
        SetMetadata(metadata)
        SetMaxTransactionFee(hedera.NewHbar(30)). //Change the default max
transaction fee
        FreezeWith(client)

```

```

if err != nil {
    panic(err)
}

```

```

//Sign with the admin private key of the token, sign with the token treasury
private key, sign with the client operator private key and submit the
transaction to a Hedera network
txResponse, err :=
tokenCreateTransaction.Sign(adminKey).Sign(treasuryKey).Execute(client)

```

```

if err != nil {
    panic(err)
}

```

```

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

```

```

//Get the token ID from the receipt
tokenId := receipt.TokenID

```

```

fmt.Printf("The new token ID is %v\n", tokenId)

```

```

//v2.1.0

```

```

{% endtab %}
{% endtabs %}

```

```

[^1]:

```

```

[^2]:

```

```

# delete-a-token.md:

```

```

Delete a token

```

Deleting a token marks a token as deleted, though it will remain in the ledger. The operation must be signed by the specified Admin Key of the Token. If the Admin Key is not set, the Transaction will result in TOKEN\IS\IMMUTABLE. Once deleted update, mint, burn, wipe, freeze, unfreeze, grant KYC, revoke KYC and token transfer transactions will resolve to TOKEN\WAS\DELETED.

NFTs

You cannot delete a specific NFT. You can delete the class of the NFT specified by the token ID after you have burned all associated NFTs associated with the token class

Transaction Signing Requirements

Admin key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	
-----	-----	
setTokenId(<tokenId>)	TokenId	The ID of the token to delete Required

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
TokenDeleteTransaction transaction = new TokenDeleteTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the admin private key of the
account, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(adminKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze the unsigned transaction for manual signing
const transaction = await new TokenDeleteTransaction()
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the admin private key of the token
const signTx = await transaction.sign(adminKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}
```

```

{% tab title="Go" %}
go
//Create the transaction and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenDeleteTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin private key of the account, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(adminKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

disable-kyc-account-flag.md:

Disable KYC account flag

Revokes the KYC flag to the Hedera account for the given Hedera token. This transaction must be signed by the token's KYC Key. If this key is not set, you can submit a TokenUpdateTransaction to provide the token with this key.

If the provided account is not found, the transaction will be resolved to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If the provided token is not found, the transaction will resolve to INVALIDTOKENID.

If the provided token has been deleted, the transaction will resolve to TOKENWASDELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKENHASNOKYCKEY. Once executed, the Account is marked as KYC Revoked

Transaction Signing Requirements

KYC key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	

setTokenId(<tokenId>)	TokenId	The token ID that is associated with the account to remove the KYC flag for
Required		
setAccountId(<setAccountId>)	AccountId	The account ID that is associated with the account to remove the KYC flag
Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Remove the KYC flag from an account
TokenRevokeKycTransaction transaction = new TokenRevokeKycTransaction()
    .setTokenId(tokenId)
    .setAccountId(accountId);

//Freeze the unsigned transaction, sign with the kyc private key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(kycKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//Version: 2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Remove the KYC flag on account and freeze the transaction for signing
const transaction = await new TokenRevokeKycTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the kyc private key of the token
const signTx = await transaction.sign(kycKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5
```

```

{% endtab %}

{% tab title="Go" %}
go
//Remove the KYC flag from an account and freeze the transaction for signing
transaction, err = hedera.NewTokenRevokeKycTransaction().
    SetTokenID(tokenId).
    SetAccountID(accountId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the kyc private key of the token, submit the transaction to a Hedera
network
txResponse, err := transaction.Sign(kycKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

dissociate-tokens-from-an-account.md:

Dissociate tokens from an account

Dissociates the provided Hedera account from the provided Hedera tokens. This transaction must be signed by the provided account's key. Once the association is removed, no token-related operation can be performed to that account. AccountBalanceQuery and AccountInfoQuery will not return anything related to the dissociated token.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If any of the provided tokens is not found, the transaction will resolve to INVALIDTOKENREF.

If an association between the provided account and any of the tokens does not exist, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If the provided account has a nonzero balance with any of the provided tokens, the transaction will resolve to TRANSACTIONREQUIRESZEROTOKENBALANCES.

On success, associations between the provided account and tokens are removed.

```

{% hint style="info" %}

```

The account is required to have a zero balance of the token you wish to dissociate. If a token balance is present, you will receive a TRANSACTIONREQUIRESZEROTOKENBALANCES error.

```
{% endhint %}
```

Transaction Signing Requirements

The key of the account the token is being dissociated with
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	
-----	-----	
setTokenIds(<tokenId>)	TokenId	The tokens to be dissociated with the provided account Required
setAccountId(<accountId>)	AccountId	The account to be dissociated with the provided tokens Required

```
{% tabs %}
{% tab title="Java" %}
java
//Dissociate a token from an account
TokenDissociateTransaction transaction = new TokenDissociateTransaction()
    .setAccountId(accountId)
    .setTokenIds(tokenId);

//Freeze the unsigned transaction, sign with the private key of the account that
is being dissociated from a token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(accountKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Dissociate a token from an account and freeze the unsigned transaction for
signing
const transaction = await new TokenDissociateTransaction()
    .setAccountId(accountId)
    .setTokenIds([tokenId])
    .freezeWith(client);

//Sign with the private key of the account that is being associated to a token
const signTx = await transaction.sign(accountKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);
```

```

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Dissociate the token from an account and freeze the unsigned transaction for
signing
transaction, err := hedera.NewTokenDissociateTransaction().
    SetAccountID(accountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the private key of the account that is being associated to a token,
submit the transaction to a Hedera network
txResponse, err = transaction.Sign(accountKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

enable-kyc-account-flag.md:

Enable KYC account flag

Grants KYC to the Hedera accounts for the given Hedera token. This transaction must be signed by the token's KYC Key.

If the provided account is not found, the transaction will be resolved to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED.

If the provided token is not found, the transaction will resolve to

INVALIDTOKENID.

If the provided token has been deleted, the transaction will resolve to TOKENWASDELETED.

If an Association between the provided token and the account is not found, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If no KYC Key is defined, the transaction will resolve to TOKEN\HAS\NO\KYC\KEY. Once executed, the Account is marked as KYC Granted.

Transaction Signing Requirements

KYC key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate your transaction fee cost.

Methods

Method	Type	Description
Required		
----- -----		
----- -----		
setTokenId(<tokenId>)	TokenId	The token for this account to have passed KYC Required
setAccountId(<accountId>)	AccountId	The account for this token to have passed KYC Required

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Enable KYC flag on account
```

```
TokenGrantKycTransaction transaction = new TokenGrantKycTransaction()
```

```
    .setAccountId(accountId)
```

```
    .setTokenId(tokenId);
```

```
//Freeze the unsigned transaction, sign with the kyc private key of the token,  
submit the transaction to a Hedera network
```

```
TransactionResponse txResponse =
```

```
transaction.freezeWith(client).sign(kycKey).execute(client);
```

```
//Request the receipt of the transaction
```

```
TransactionReceipt receipt = txResponse.getReceipt(client);
```

```
//Obtain the transaction consensus status
```

```
Status transactionStatus = receipt.status;
```

```
System.out.println("The transaction consensus status is " +transactionStatus);
```

```
//v2.0.1
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Enable KYC flag on account and freeze the transaction for manual signing
```

```
const transaction = await new TokenGrantKycTransaction()
```

```
    .setAccountId(accountId)
```

```
    .setTokenId(tokenId)
```

```
    .freezeWith(client);
```

```
//Sign with the kyc private key of the token
```

```
const signTx = await transaction.sign(kycKey);
```

```

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Enable KYC flag on account and freeze the transaction for manual signing
transaction, err = hedera.NewTokenGrantKycTransaction().
    SetAccountID(accountId).
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the kyc private key of the token, submit the transaction to a Hedera
network
txResponse, err := transaction.Sign(kycKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

errors.md:

Network Response Messages

Network response messages and their descriptions.

Network Response	Description

-----	-----

----- ACCOUNTFROZENFORTOKEN and cannot transact with the token	The account is frozen
 TOKENSPERACCOUNTLIMITEXCEEDED token relations for a given account is exceeded	The maximum number of
 INVALIDTOKENID or does not exist	The token is invalid
 INVALIDTOKENDECIMALS decimals	Invalid token
 INVALIDTOKENINITIALSUPPLY supply	Invalid token initial
 INVALIDTREASURYACCOUNTFORTOKEN not exist or is deleted	Treasury account does
 INVALIDTOKENSYMBOL UTF-8 capitalized alphabetical string	Token Symbol is not
 TOKENHASNOFREEZEKEY on a token	Freeze key is not set
 TRANSFERSNOTZEROSUMFORTOKEN list are not net-zero	Amounts in the transfer
 MISSINGTOKENSYMBOL provided	Token Symbol is not
 TOKENSYMBOLTOOLONG long	Token Symbol is too
 ACCOUNTKYCNOTGRANTEDFORTOKEN the account does not have KYC granted	KYC must be granted and
 TOKENHASNOKYCKEY a token	KYC key is not set on
 INSUFFICIENTTOKENBALANCE sufficient for the transaction	Token balance is not
 TOKENWASDELETED cannot be executed on deleted token	Token transactions
 TOKENHASNOSUPPLYKEY set on a token	The supply key is not
 TOKENHASNOWIPEKEY set on a token	The wipe key is not
 INVALIDTOKENMINTAMOUNT	Invalid mint amount
 INVALIDTOKENBURNAMOUNT	Invalid burn amount
 TOKENNOTASSOCIATEDTOACCOUNT associated with an account	Account has not been
 CANNOTWIPETOKENTREASURYACCOUNT operation on treasury account	Cannot execute wipe
 INVALIDKYCKEY	Invalid kyc key
 INVALIDWIPEKEY	Invalid wipe key

INVALIDFREEZEKEY	Invalid freeze key
INVALIDSUPPLYKEY	Invalid supply key
MISSINGTOKENNAME provided	Token Name is not
TOKENNAMETOOLONG long	Token Name is too
INVALIDWIPINGAMOUNT amount must not be negative, zero or bigger than the token holder balance	The provided wipe
TOKENISIMMUTABLE have Admin key set, thus update/delete transactions cannot be performed	The token does not
TOKENALREADYASSOCIATEDTOACCOUNT operation specified a token already associated with the account	An associateToken
TRANSACTIONREQUIRESZEROTOKENBALANCES is invalid until all token balances for the target account are zero	An attempted operation
ACCOUNTISTREASURY operation is invalid because the account is a treasury	An attempted
TOKENIDREPEATEDINTOKENLIST in the token list	Same TokenIDs present
TOKENTRANSFERLISTSIZELIMITEXCEEDED token transfers (both from and to) allowed for token transfer list	Exceeded the number of
EMPTYTOKENTRANSFERBODY TokenTransfersTransactionBody has no TokenTransferList	
EMPTYTOKENTRANSFERACCOUNTAMOUNTS TokenTransfersTransactionBody has a TokenTransferList with no AccountAmounts	
FRACTIONDIVIDESBYZERO fee set a denominator of zero	A custom fractional
INSUFFICIENTPAYERBALANCEFORCUSTOMFEE could not afford a custom fee	The transaction payer
CUSTOMFEESLISTTOOLONG longer than allowed limit 10	The customFees list is
INVALIDCUSTOMFEECOLLECTOR feeCollector accounts for customFees is invalid	Any of the
INVALIDTOKENIDINCUSTOMFEES customFees is invalid	Any of the token Ids in
TOKENNOTASSOCIATEDTOFEECOLLECTOR customFees are not associated to feeCollector	Any of the token Ids in
CUSTOMFEEENOTFULLYSPECIFIED entry did not specify either a fixed or fractional fee	A custom fee schedule
CUSTOMFEEMUSTBEPOSITIVE be assessed at this time	Only positive fees may
TOKENHASNOFEESCHEDULEKEY set on token	Fee schedule key is not
CUSTOMFEEOUTSIDENUMERICRANGE fee exceeded the range of a 64-bit signed integer	A fractional custom

INVALIDCUSTOMFRACTIONALFEESUM	The sum of all custom
fractional fees must be strictly less than 1	
FRACTIONALFEEMAXAMOUNTLESSTHANMINAMOUNT	Each fractional custom
fee must have its maximum\amount, if specified, at least its minimum\amount	
CUSTOMSCHEDULEALREADYHASNOFEES	A fee schedule update
tried to clear the custom fees from a token whose fee schedule was already empty	
CUSTOMFEEDENOMINATIONMUSTBEFUNGIBLECOMMON	Only tokens of type
FUNGIBLE\COMMON can be used as fee schedule denominations	
CUSTOMFRACTIONALFEEONLYALLOWEDFORFUNGIBLECOMMON	Only tokens of type
FUNGIBLE\COMMON can have fractional fees	
INVALIDCUSTOMFEESCHEDULEKEY	The provided custom
fee schedule key was invalid	
ACCOUNTAMOUNTTRANSFERONLYALLOWEDFORFUNGIBLECOMMON	An AccountAmount token
transfers list referenced a token type other than FUNGIBLE\COMMON	
INVALIDTOKENMINTMETADATA	The requested token
mint metadata was invalid	
INVALIDTOKENBURNMETADATA	The requested token
burn metadata was invalid	
PAYERACCOUNTDELETED	The payer account
has been marked as deleted	
CUSTOMFEECHARGINGEXCEEDEDMAXRECURSIONDEPTH	The reference chain of
custom fees for a transferred token exceeded the maximum length of 2	
CUSTOMFEECHARGINGEXCEEDEDMAXACCOUNTAMOUNTS	More than 20 balance
adjustments were to satisfy a CryptoTransfer and its implied custom fee payments	
INSUFFICIENTSENDERACCOUNTBALANCEFORCUSTOMFEE	The sender account in
the token transfer transaction could not afford a custom fee	
SERIALNUMBERLIMITREACHED	Currently no more
than 4,294,967,295 NFTs may be minted for a given unique token type	
CUSTOMROYALTYFEEONLYALLOWEDFORNONFUNGIBLEUNIQUE	Only tokens of type NON\
FUNGIBLE\UNIQUE can have royalty fees	
TOKENISPAUSED	Token is paused.
This Token cannot be a part of any kind of Transaction until unpaused.	
TOKENHASNOPAUSEKEY	Pause key is not set
on token	
INVALIDPAUSEKEY	The provided pause
key was invalid	

freeze-an-account.md:

Freeze an account

Freezes transfers of the specified token for the account. The transaction must be signed by the token's Freeze Key.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID. If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\

TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Frozen and will not be able to receive or send tokens unless unfrozen.

The operation is idempotent

Transaction Signing Requirements

Freeze key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	-----
setTokenId(<tokenId>)	TokenId	The token for this account to be frozen
Required		
setAccountId(<accountId>)	AccountId	The account to be frozen
Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Freeze an account from transferring a token
TokenFreezeTransaction transaction = new TokenFreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the sender freeze private key of
the token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(freezeKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.print("The transaction consensus status is " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Freeze an account from transferring a token
const transaction = await new TokenFreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);
```

```

//Sign with the freeze key of the token
const signTx = await transaction.sign(freezeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Freeze an account from transferring a token
transaction, err = hedera.NewTokenFreezeTransaction().
    SetAccountID(accountId).
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the freeze private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(freezeKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

get-account-token-balance.md:

Get account token balance

To get the balance of tokens for an account, you can submit an account balance query. The account balance query will return the tokens the account holds in a list format.

Query Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

Method	Type	Requirement
-----	-----	-----
setAccountId(<accountId>)	AccountId	Required

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
AccountBalanceQuery query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Sign with the operator private key and submit to a Hedera network
AccountBalance tokenBalance = query.execute(client);

System.out.println("The token balance(s) for this account: "
+tokenBalance.tokens);

//v2.0.9

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new AccountBalanceQuery()
    .setAccountId(accountId);

//Sign with the client operator private key and submit to a Hedera network
const tokenBalance = await query.execute(client);

console.log("The token balance(s) for this account: "
+tokenBalance.tokens.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the query
query := hedera.NewAccountBalanceQuery().
    SetAccountID(accountId)

//Sign with the client operator private key and submit to a Hedera network
tokenBalance, err := query.Execute(client)

if err != nil {
    panic(err)
}

fmt.Printf("The token balance(s) for this account: %v\n", tokenBalance)

//v2.1.0

{% endtab %}
{% endtabs %}

# get-nft-token-info.md:
```

Get NFT info

A query that returns information about a non-fungible token (NFT). You request the info for an NFT by specifying the NFT ID.

Token Allowances

Only when a spender is set on an explicit NFT ID of a token, we return the spender ID in the `TokenNftInfoQuery` for the respective NFT. If `approveTokenNftAllowanceAllSerials` is used to approve all NFTs for a given token class and no NFT ID is specified, we will not return a spender ID for all the serial numbers of that token.

Query Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your query fee cost

```
{% hint style="warning" %}  
Requesting NFT info by Token ID or Account ID is deprecated.  
{% endhint %}
```

The request returns the following information:

Item	Description
-----	-----
NFT ID	The ID of the non-fungible token in x.y.z format.
Account ID	The account ID of the current owner of the NFT
Creation Time	The effective consensus timestamp at which the NFT was minted
Metadata	Represents the unique metadata of the NFT
Ledger ID	The ID of the network (mainnet, testnet, previewnet).
Reference HIP-198.	
Spender ID	The spender account ID for the NFT. This is only returned if the NFT ID was specifically approved.

Methods

```
<table><thead><tr><th width="235">Method</th><th width="75">Type</th><th  
width="308">Description</th><th>Requirement</th></tr></thead><tbody><tr><td><code>  
setNftId(&#x3C;nftId>)</code></td><td><a  
href="nft-id.md">NftId</a></td><td>Applicable only to tokens of type  
<code>NONFUNGIBLEUNIQUE</code>. Gets info on a NFT for a given TokenID (of type  
<code>NONFUNGIBLEUNIQUE</code>) and serial  
number.</td><td>Optional</td></tr></tbody></table>
```

```
{% tabs %}  
{% tab title="Java" %}  
java  
//Returns the info for the specified NFT ID  
List<TokenNftInfo> nftInfos = new TokenNftInfoQuery()  
    .setNftId(nftId)  
    .execute(client);
```

```
//v2.0.14
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Returns the info for the specified NFT ID
const nftInfos = await new TokenNftInfoQuery()
    .setNftId(nftId)
    .execute(client);
```

```
//v2.0.28
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
//Returns the info for the specified NFT ID
nftInfo, err := NewTokenNftInfoQuery().
    SetNftID(nftID).
    Execute(client)
```

```
//v2.1.16
```

```
{% endtab %}
{% endtabs %}
```

get-token-info.md:

Get token info

Gets information about a fungible or non-fungible token instance.

Query Fees

Please see the transaction and query fees table for the base transaction fee.
Please use the Hedera fee estimator to estimate the cost of your query fee.

The token info query returns the following information:

Item	Description
Token ID	ID of the token instance
Token Type	The type of token (fungible or non-fungible)
Name	The name of the token. It is a string of ASCII only characters
Symbol	The symbol of the token. It is a UTF-8 capitalized alphabetical string
Decimals	The number of decimal places a token is divisible by
Total Supply	The total supply of tokens that are currently in circulation
Treasury	The ID of the account which is set as Treasury
Custom Fees	The custom fee schedule of the token, if any
Fee Schedule Key	Fee schedule key, if any
Admin Key	The key which can perform update/delete operations on the token. If empty, the token can be perceived as immutable (not being able to be updated/deleted)
KYC Key	The key which can grant or revoke KYC of an account for the token's transactions. If empty, KYC is not required, and KYC grant or revoke operations are not possible.
Freeze Key	The key which can freeze or unfreeze an account for token transactions. If empty, freezing is not possible
Wipe Key	The key which can wipe token balance of an account. If empty, wipe is not possible
Supply Key	The key which can

change the supply of a token. The key is used to sign Token Mint/Burn operations	
Pause Key	The key that can pause or unpause the token from participating in transactions.
Pause Status	Whether or not the token is paused. false = not paused true = paused
Max Supply	The max supply of the token
Supply Type	The supply type of the token
Default Freeze Status	The default Freeze status (not applicable = null, frozen = false, or unfrozen = true) of Hedera accounts relative to this token. FreezeNotApplicable is returned if Token Freeze Key is empty. Frozen is returned if Token Freeze Key is set and defaultFreeze is set to true. Unfrozen is returned if Token Freeze Key is set and defaultFreeze is set to false. FreezeNotApplicable = null; Frozen = true; Unfrozen = false;
Default KYC Status	The default KYC status (KycNotApplicable or Revoked) of Hedera accounts relative to this token. KycNotApplicable is returned if KYC key is not set, otherwise Revoked. KycNotApplicable = null; Granted = false; Revoked = true;
Auto Renew Account	An account which will be automatically charged to renew the token's expiration, at autoRenewPeriod interval
Auto Renew Period	The interval at which the auto-renew account will be charged to extend the token's expiry
Expiry	The epoch second at which the token will expire; if an auto-renew account and period are specified, this is coerced to the current epoch second plus the autoRenewPeriod
Ledger ID	The ID of the network the response came from. See HIP-198
Memo	Short publicly visible memo about the token, if any
Metadata Key	The key which can change the metadata of a token definition or individual NFT.
Metadata	The metadata for the token.

Methods

Method	Type	Requirement
<code>setTokenId(<tokenId>)</code>	TokenId	Required
<code><tokenId>.tokenId</code>	Optional	Optional
<code><tokenId>.name</code>	Optional	Optional
<code><tokenId>.symbol</code>	String	Optional
<code><tokenId>.decimals</code>	Optional	Optional
<code><tokenId>.customFees</code>	List<CustomFee>	Optional
<code><tokenId>.totalSupply</code>	long	Optional
<code><tokenId>.treasuryAccountId</code>	AccountId	Optional
<code><tokenId>.adminKey</code>	Key	Optional
<code><tokenId>.kycKey</code>	Key	Optional
<code><tokenId>.freezeKey</code>	Key	Optional
<code><tokenId>.feeScheduleKey</code>	Key	Optional
<code><tokenId>.wipeKey</code>	Key	Optional
<code><tokenId>.supplyKey</code>	Key	Optional
<code><tokenId>.defaultFreezeStatus</code>	boolean	Optional
<code><tokenId>.defaultKycStatus</code>	boolean	Optional
<code><tokenId>.isDeleted</code>	boolean	Optional
<code><tokenId>.tokenType</code>		

TokenType	Optional
<code>&#x3C;TokenInfo>.supplyType</code>	TokenSupplyType
Optional	
<code>&#x3C;TokenInfo>.maxSupply</code>	
long	Optional
<code>&#x3C;TokenInfo>.pauseKey</code>	
Key	Optional
<code>&#x3C;TokenInfo>.pauseStatus</code>	boolean
Optional	
<code>&#x3C;TokenInfo>.autoRenewAccount</code>	
AccountId	Optional
<code>&#x3C;TokenInfo>.autoRenewPeriod</code>	Duration
Optional	
<code>&#x3C;TokenInfo>.ledgerId</code>	
LedgerId	Optional
<code>&#x3C;TokenInfo>.expiry</code>	Instant
Optional	
<code>&#x3C;TokenInfo>.metadata</code>	
bytes	Optional

```
{% tabs %}
{% tab title="Java" %}
java
//Create the query
TokenInfoQuery query = new TokenInfoQuery()
    .setTokenId(newTokenId);

//Sign with the client operator private key, submit the query to the network and
get the token supply
long tokenSupply = query.execute(client).totalSupply;

System.out.println("The token info is " +tokenSupply);

//v2.0.14

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the query
const query = new TokenInfoQuery()
    .setTokenId(newTokenId);

//Sign with the client operator private key, submit the query to the network and
get the token supply
const tokenSupply = (await query.execute(client)).totalSupply;

console.log("The total supply of this token is " +tokenSupply);

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Create the query
query := hedera.NewTokenInfoQuery().
    SetTokenID(tokenId)

//Sign with the client operator private key and submit to a Hedera network
tokenInfo, err := query.Execute(client)

if err != nil {
    panic(err)
}

fmt.Printf("The token info is %v\n", tokenInfo)

//v2.1.0
```



```
{% endtab %}  
{% endtabs %}
```

mint-a-token.md:

Mint a token

Minting fungible token allows you to increase the total supply of the token. Minting a non-fungible token creates an NFT with its unique metadata for the class of NFTs defined by the token ID. The Supply Key must sign the transaction.

If no Supply Key is defined, the transaction will resolve to TOKEN\HAS\NO\SUPPLY\KEY. The maximum total supply a token can have is $2^{63}-1$.

The amount provided must be in the lowest denomination possible.

Example: Token A has 2 decimals. In order to mint 100 tokens, one must provide an amount of 10000. In order to mint 100.55 tokens, one must provide an amount of 10055.

The metadata field is specific to NFTs. Once an NFT is minted, the metadata cannot be changed and is immutable.

You can use the metadata field to add a URI that contains additional information about the token. You can view the metadata schema [here](#). The metadata field has a 100-character limit.

The serial number for the NFT is returned in the receipt of the transaction.

When minting NFTs, do not set the amount. The amount is used for minting fungible tokens only.

This transaction accepts zero unit minting operations for fungible tokens (HIP-564)

Transaction Signing Requirements

Supply key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate your transaction fee cost.

Methods

Method	Type	Description	Requirement
<code>setTokenId(&#x3C;tokenId>)</code>	TokenId	The token ID for which to mint additional tokens	Required
<code>setAmount(&#x3C;amount>)</code>	long	Applicable to tokens of type <code>FUNGIBLECOMMON</code> . The amount to mint to the Treasury Account. The amount must be a positive non-zero number represented in the lowest denomination of the token. The new supply must be lower than <code>2⁶³-1</code> .	Optional
<code>setMetadata(&#x3C;metaData>)</code>	List<byte[]>	Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code> . A list of metadata that are being created. The maximum allowed size of each metadata is 100 bytes and is immutable.	Optional
<code>addMetadata(&#x3C;metaData>)</code>	byte []	Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code> . A list of metadata that are being created. The maximum allowed size of each metadata is 100 bytes and is immutable.	Optional

```
{% tabs %}  
{% tab title="Java" %}  
java
```

```

//Mint another 1,000 tokens
TokenMintTransaction transaction = new TokenMintTransaction()
    .setTokenId(tokenId)
    .setMaxTransactionFee(new Hbar(20)) //Use when HBAR is under 10 cents
    .setAmount(1000);

//Freeze the unsigned transaction, sign with the supply private key of the
token, submit the transaction to a Hedera network
TransactionResponse txResponse = transaction
    .freezeWith(client)
    .sign(supplyKey)
    .execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus;

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Mint another 1,000 tokens and freeze the unsigned transaction for manual
signing
const transaction = await new TokenMintTransaction()
    .setTokenId(tokenId)
    .setAmount(1000)
    .setMaxTransactionFee(new Hbar(20)) //Use when HBAR is under 10 cents
    .freezeWith(client);

//Sign with the supply private key of the token
const signTx = await transaction.sign(supplyKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Mint another 1,000 tokens and freeze the unsigned transaction for manual
signing
transaction, err = hedera.NewTokenMintTransaction().
    SetTokenID(tokenId).
    SetAmount(1000).
    //Use when HBAR is under 10 cents
    SetMaxTransactionFee(hedera.HbarFrom(20, hedera.HbarUnits.Hbar)).
    FreezeWith(client)

```

```

if err != nil {
    panic(err)
}

//Sign with the supply private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.
    Sign(supplyKey).
    Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

# nft-id.md:

NFT ID

The ID of a non-fungible token (NFT). The NFT ID is composed of the token ID and
a serial number.

| Constructor | Description |
| ----- | ----- |
| new NftId(<tokenId>,<serial>) | Initializes the NftId object |

java
new NftId()

Methods

| Method | Type | Requirement |
| ----- | ----- | ----- |
| NftId.fromString(<id>) | String | Optional |
| NftId.fromBytes(<id>) | bytes \[] | Optional |

{% tabs %}
{% tab title="Java" %}
java
new NftId(new TokenId(0,0,2), 56562);

// v2.0.11

{% endtab %}

{% tab title="JavaScript" %}

```

```

javascript
new NftId(new TokenId(0,0,2), 56562);

// v2.0.28

{% endtab %}

{% tab title="Go" %}
java
nftId := hedera.NftID{
    TokenID: tokenId,
    SerialNumber: serialNum,
}
// v2.1.13

{% endtab %}
{% endtabs %}

```

pause-a-token.md:

Pause a token

A token pause transaction prevents the token from being involved in any kind of operation. The token's pause key is required to sign the transaction. This is a key that is specified during the creation of a token. If a token has no pause key, you will not be able to pause the token. If the pause key was not set during the creation of a token, you will not be able to update the token to add this key.

The following operations cannot be performed when a token is paused and will result in a TOKENISPAUSED status.

- Updating the token
- Transferring the token
- Transferring any other token where it has its paused key in a custom fee schedule
- Deleting the token
- Minting or burning a token
- Freezing or unfreezing an account that holds the token
- Enabling or disabling KYC
- Associating or disassociating a token
- Wiping a token

Once a token is paused, token status will update to paused. To verify if the token's status has been updated to paused, you can request the token info via the SDK or use the token info mirror node query. If the token is not paused the token status will be unpaused. The token status for tokens that do not have an assigned pause key will state PauseNotApplicable.

Transaction Signing Requirements

- The pause key of the token
- Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description	Requirement
--------	------	-------------	-------------

```

| ----- | ----- | ----- | -----
| setTokenId(<tokenId>) | TokenId | The ID of the token to pause | Required |

{% tabs %}
{% tab title="Java" %}
java
//Create the token pause transaction and specify the token to pause
TokenPauseTransaction transaction = new TokenPauseTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the pause key, submit the
transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(pauseKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.2.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the token pause transaction, specify the token to pause, freeze the
unsigned transaction for signing
const transaction = new TokenPauseTransaction()
    .setTokenId(tokenId);
    .freezeWith(client);

//Sign with the pause key
const signTx = await transaction.sign(pauseKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.2.0

{% endtab %}

{% tab title="Go" %}
go
//Create the token pause transaction, specify the token to pause, freeze the
unsigned transaction for signing
transaction, err := hedera.NewTokenPauseTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

```

```

}

//Sign with the pause key
txResponse, err = transaction.Sign(pauseKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.3.0

{% endtab %}
{% endtabs %}

```

README.md:

Token Service

reject-an-airdrop.md:

Reject an airdrop

The TokenRejectTransaction allows users to reject and return unwanted airdrops to the treasury account without incurring custom fees. This transaction type supports rejecting the full balance of fungible tokens or individual NFT serial numbers with support for up to 10 token rejections in a single transaction. Rejection is not supported if the token has been frozen or paused. Note that the transaction does not dissociate an account from the token. For dissociation, one must perform a TokenDissociateTransaction.

Important Notes:

When a token is rejected, usedautoassociations is not decremented. This field tracks the total number of auto-associations made, not the current number of token types associated with the account.

The receiversigrequired setting is ignored on the treasury account when handling a TokenReject. Rejection will always proceed without custom fees.

Token rejection is not supported if the token is frozen or paused.

Transaction Signing Requirements

The key of the account rejecting the tokens
The transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee. Please use the Hedera fee estimator to estimate the cost of your transaction fee.

Methods

Method	Type	Description
<code>setOwnerId(accountId)</code>	AccountId	An account holding the tokens to be rejected.
<code>addTokenId(&#x3C;tokenId>)</code>	List<TokenId>	A list of one or more token IDs to be rejected.
<code>addNftId(&#x3C;nftId>)</code>	List<NftId>	A single specific serialized non-fungible/unique token.

```
{% tabs %}
{% tab title="Java" %}
java
// Create the token reject transaction
TokenRejectTransaction transaction = new TokenRejectTransaction()
    .setOwnerId(accountId)
    .addTokenId(tokenId)
    .freezeWith(client);

// Sign with the account Id key and submit the transaction to a Hedera network
TransactionResponse txResponse = transaction.sign(accountIdKey).execute(client);

// Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

// Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create the token reject transaction
const transaction = new TokenRejectTransaction()
    .setOwnerId(accountId)
    .addTokenId(tokenId)
    .freezeWith(client);

// Sign with the account Id key and submit the transaction to a Hedera network
const txResponse = await transaction.sign(accountIdKey).execute(client);

// Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

// Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

{% endtab %}

{% tab title="Go" %}
go
// Create the token reject transaction
transaction := hedera.NewTokenRejectTransaction().
    SetAccountID(accountId).
```

```

        AddTokenIDs(tokenIds).
        FreezeWith(client)

if err != nil {
    panic(err)
}

// Sign with the account Id key and submit the transaction to a Hedera network
txResponse, err := transaction.
    Sign(accountIdKey).
    Execute(client)

if err != nil {
    panic(err)
}

// Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

// Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

{% endtab %}
{% endtabs %}

```

token-id.md:

Token ID

Constructs a TokenId.

Constructor	Description
new TokenId(<shard>,<realm>,<token>)	Initializes the TokenId object

```

java
new TokenId()

```

Methods

Method	Type	Description
TokenId.fromString(<tokenId>)	String	Constructs a token ID from a String value
TokenId.fromSolidityAddress(<address>)	String	Constructs a token ID from a solidity address
TokenId.fromBytes(<bytes>)	byte\[]	Constructs a token ID from bytes

```

{% tabs %}
{% tab title="Java" %}
java
TokenId tokenId = new TokenId(0,0,5);
System.out.println(tokenId);

```



```
TokenId tokenIdFromString = TokenId.fromString("0.0.3");
System.out.println(tokenIdFromString);
```

```
//Version: 2.0.1
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
const tokenId = new TokenId(0,0,5);
console.log(tokenId.toString());
```

```
const tokenIdFromString = TokenId.fromString("0.0.3");
console.log(tokenIdFromString.toString());
```

```
//Version 2.0.7
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
tokenId := hedera.TokenID {
    Shard: 0,
    Realm: 0,
    Token: 5,
}
```

```
//v2.1.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
# token-types.md:
```

```
Token types
```

There are two types of tokens you can create using the Hedera Token Service: fungible and non-fungible tokens. A fungible (FUNGIBLECOMMON) token is a class of tokens that can be interchangeable with another in the same class. Tokens in this class share the same value and share all the same properties. A non-fungible token (NONFUNGIBLEUNIQUE) is a class of tokens that are not identical to the other tokens in the same class. This token type cannot be interchanged with other tokens and is differentiated by serial numbers that reference each unique token. The SDKs default to creating fungible tokens if the token type during creation is not specified.

```
Token Type
```

```
FUNGIBLE
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
TokenType.FUNGIBLECOMMON
```

```
// v2.0.11
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
TokenType.FungibleCommon
```

```
// v2.0.28

{% endtab %}

{% tab title="Go" %}
go
hedera.TokenTypeFungibleCommon

// v2.1.14

{% endtab %}
{% endtabs %}

NON-FUNGIBLE

{% tabs %}
{% tab title="Java" %}
java
TokenType.NONFUNGIBLEUNIQUE

// v2.0.11

{% endtab %}

{% tab title="JavaScript" %}
javascript
TokenType.NonFungibleUnique

// v2.0.28

{% endtab %}

{% tab title="Go" %}
go
hedera.TokenTypeNonFungibleUnique

// v2.1.14

{% endtab %}
{% endtabs %}
```

transfer-tokens.md:

Transfer tokens

Transfer tokens from some accounts to other accounts. The transaction must be signed by the sending account. Each negative amount is withdrawn from the corresponding account (a sender), and each positive one is added to the corresponding account (a receiver). All amounts must have a sum of zero. This does not apply to NFT token transfers. Each amount is a number with the lowest denomination possible for a token. Example: Token X has 2 decimals. Account A transfers an amount of 100 tokens by providing 10000 as the amount in the TransferList. If Account A wants to send 100.55 tokens, he must provide 10055 as the amount. If any sender account fails to have a sufficient token balance, then the entire transaction fails and none of the transfers occur, though the transaction fee is still charged. This transaction accepts zero unit token transfer operations for fungible tokens (HIP-564).

Custom Fee Tokens

Custom fee tokens are tokens that have a unique custom fee schedule associated to them. The sender account is required to pay for the custom fee(s) associated

with the token that is being transferred. The sender account must have the amount of the custom fee token being transferred and the custom fee amounts to successfully process the transaction. You can check to see if the token has a custom fee schedule by requesting the token info query. Token with custom fees allow up to two levels of nesting in a transfer transaction.

Transaction Signing Requirements

Transaction Fees

Methods

```
| <p><code>addApprovedTokenTransfer(&#x3C;tokenId>, &#x3C;accountId>,
&#x3C;value>)</code>(previewnet)<br></p> | TokenId, AccountId,
long | <p>The owner account ID and token the spender
is authorized to transfer from. The debiting account is the owner
account.<br>Applicable to allowance transfers only.<br></p>
|
| <p><code>addApprovedTokenTransferWithDecimals(&#x3C;tokenId>,
&#x3C;accountId>, &#x3C;value>, &#x3C;decimals>)</code><br></p> | TokenId,
AccountId, long, int | <p>The owner account ID and token
ID (with decimals) the spender is authorized to transfer from. The debit account
is the account ID of the sender.<br>Applicable to allowance transfers only.</p>
|
| <p><code>addApprovedNftTransfer(&#x3C;nftId>, &#x3C;sender>,
&#x3C;receiver>)</code><br></p> | NftId,
AccountId, AccountId | <p>The NFT ID the spender is authorized to transfer. The
sender is the owner account and receiver is the receiving account.<br>Applicable
to allowance transfers only.</p>
|
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transfer transaction
TransferTransaction transaction = new TransferTransaction()
    .addTokenTransfer(tokenId, OPERATORID, -1)
    .addTokenTransfer(tokenId, accountId, 1);

//Sign with the client operator key and submit the transaction to a Hedera
network
TransactionResponse txResponse = transaction.execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transfer transaction
const transaction = await new TransferTransaction()
    .addTokenTransfer(tokenId, accountId1, -1)
    .addTokenTransfer(tokenId, accountId2, 1)
    .freezeWith(client);

//Sign with the sender account private key
const signTx = await transaction.sign(accountKey1);

//Sign with the client operator private key and submit to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());
```

```
//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transfer transaction and freeze the transaction from further
modification
transaction, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId, accountId1, -1).
    AddTokenTransfer(tokenId, accountId2, 0).
    FreezeWith(client)

//Sign with the accountId1 private key, sign with the client operator key and
submit to a Hedera network
txResponse, err := transaction.Sign(accountKey1).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)
//v2.1.0

{% endtab %}
{% endtabs %}
```

unfreeze-an-account.md:

Unfreeze an account

Unfreezes transfers of the specified token for the account. The transaction must be signed by the token's Freeze Key.

If the provided account is not found, the transaction will resolve to INVALID\ACCOUNT\ID.

If the provided account has been deleted, the transaction will resolve to ACCOUNT\DELETED.

If the provided token is not found, the transaction will resolve to INVALID\TOKEN\ID.

If the provided token has been deleted, the transaction will resolve to TOKEN\WAS\DELETED.

If an Association between the provided token and account is not found, the transaction will resolve to TOKEN\NOT\ASSOCIATED\TO\ACCOUNT.

If no Freeze Key is defined, the transaction will resolve to TOKEN\HAS\NO\FREEZE\KEY.

Once executed the Account is marked as Unfrozen and will be able to receive or send tokens. The operation is idempotent.

Transaction Signing Requirements

Freeze key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description
Requirement		
-----	-----	
-----	-----	
setTokenId(<tokenId>)	TokenId	The token for this account to unfreeze
Required		
setAccountId(<accountId>)	AccountId	The account to unfreeze
Required		

```
{% tabs %}
{% tab title="Java" %}
java
//Unfreeze an account
TokenUnfreezeTransaction transaction = new TokenUnfreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the sender freeze private key of
the token, submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(freezeKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt8.status;

System.out.print("The transaction consensus status is " +transactionStatus);

//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Unfreeze an account and freeze the unsigned transaction for signing
const transaction = await new TokenUnfreezeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .freezeWith(client);

//Sign with the freeze private key of the token
const signTx = await transaction.sign(freezeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt8.status;

console.log("The transaction consensus status is "
+transactionStatus.toString());
```

```
//v2.0.7

{% endtab %}

{% tab title="Go" %}
go
//Unfreeze an account and freeze the unsigned transaction for signing
transaction, err = hedera.NewUnTokenFreezeTransaction().
    SetAccountID(accountId).
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the freeze private key of the token, submit the transaction to a
Hedera network
txResponse, err := transaction.Sign(freezeKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}
```

unpause-a-token.md:

Unpause a token

A token unpause transaction is a transaction that unpauses the token that was previously disabled from participating in transactions. The token's pause key is required to sign the transaction. Once the unpause transaction is submitted the token pause status is updated to unpause.

Transaction Signing Requirements:

The pause key of the token
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Method	Type	Description	Requirement

	-----	-----	-----	-----
setTokenId(<tokenId>)	tokenId	The ID of the token to pause	Required	

Methods

```
{% tabs %}
{% tab title="Java" %}
java
//Create the token unpause transaction and specify the token to pause
TokenUnpauseTransaction transaction = new TokenUnpauseTransaction()
    .setTokenId(tokenId);

//Freeze the unsigned transaction, sign with the pause key, submit the
transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(pauseKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is: " +transactionStatus);
//v2.2.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the token unpause transaction, specify the token to pause, freeze the
unsigned transaction for signing
const transaction = new TokenUnpauseTransaction()
    .setTokenId(tokenId);
    .freezeWith(client);

//Sign with the pause key
const signTx = await transaction.sign(pauseKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status;

console.log("The transaction consensus status " +transactionStatus.toString());

//v2.2.0

{% endtab %}

{% tab title="Go" %}
go
//Create the token unpause transaction, specify the token to pause, freeze the
unsigned transaction for signing
transaction, err := hedera.NewTokenUnpauseTransaction().
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
```



```

    panic(err)
}

//Sign with the pause key
txResponse, err = transaction.Sign(pauseKey).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.3.0

{% endtab %}
{% endtabs %}

```

update-a-fee-schedule.md:

Update token custom fees

Update the custom fees for a given token. If the token does not have a fee schedule, the network response returned will be CUSTOMSCHEDULEALREADYHASNOFEES. You will need to sign the transaction with the fee schedule key to update the fee schedule for the token. If you do not have a fee schedule key set for the token, you will not be able to update the fee schedule.

Transaction Signing Requirements

Fee schedule key
Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Property	Description
Fee Schedule	The new fee schedule for the token

Methods

Method	Type
Requirement	
setTokenId(<tokenId>)	TokenId
setCustomFees(<customFees>)	List<CustomFee> Optional

```

{% tabs %}
{% tab title="Java" %}

```

```

java
//Create the transaction
TokenFeeScheduleUpdateTransaction transaction = new
TokenFeeScheduleUpdateTransaction()
    .setTokenId(tokenId)
    .setCustomFees(customFee)

//Freeze the unsigned transaction, sign with the fee schedule key of the token,
submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(feeScheduleKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//Version: 2.0.9

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and freeze for manual signing
const transaction = await new TokenFeeScheduleUpdateTransaction()
    .setTokenId(tokenId)
    .setCustomFees(customFee)
    .freezeWith(client);

//Sign the transaction with the fee schedule key
const signTx = await transaction.sign(feeScheduleKey);

//Submit the signed transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Get the transaction consensus status
const transactionStatus = receipt.status.toString();

console.log("The transaction consensus status is " +transactionStatus);
//Version: 2.0.26

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze for manual signing
transaction, err := hedera.NewTokenFeeScheduleUpdateTransaction().
    SetCustomFees(customFees).
    SetTokenID(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the fee schedule key of the token, sign with the client operator
private key and submit the transaction to a Hedera network
txResponse, err := transaction.Sign(feeScheduleKey).Execute(client)

```

```

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)
//Version: 2.1.11

{% endtab %}
{% endtabs %}

```

update-a-token.md:

Update a token

A transaction that updates the properties of an existing token. The admin key must sign this transaction to update any of the token properties. With one exception. All secondary keys can sign a transaction to change themselves. The admin key can update existing keys, but cannot add new keys if they were not set during the creation of the token. If no value is given for a field, that field is left unchanged. If a TokenUpdateTx is performed we will keep returning TOKENISIMMUTABLE when it tries to change non-key properties or the expiry without an admin key.

Property	Description
Name	The new name of the token. The token name is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). Is not required to be unique.
Symbol	The new symbol of the token. The token symbol is specified as a string of UTF-8 characters in Unicode. UTF-8 encoding of this Unicode cannot contain the 0 byte (<code>NUL</code>). Is not required to be unique.
Treasury Account	The new treasury account of the token. If the provided treasury account is not existing or deleted, the response will be <code>INVALIDTREASURYACCOUNTFORTOKEN</code> . If successful, the Token balance held in the previous Treasury Account is transferred to the new one.
Admin Key	The new admin key of the token. If the token is immutable (no Admin Key was assigned during token creation), the transaction will resolve to <code>TOKENISIMMUTABLE</code> . Admin keys cannot update to add new keys that were not specified during the creation of the token.
KYC Key	The new KYC key of the token. If the token does not have currently a KYC key, the transaction will resolve to <code>TOKENHASNOKYCKEY</code> .
Freeze Key	The new freeze key of the token. If the token does not have currently a freeze key, the transaction will resolve to <code>TOKENHASNOFREEZEKEY</code> .
Fee Schedule Key	If set, the new key to use to update the token's custom fee schedule; if the token does not currently have this key, transaction will resolve to <code>TOKENHASNOFEESCHEDULEKEY</code> .
Pause Key	Update the token's existing pause key. The pause key has the ability to pause or unpause a token.
Wipe Key	The new wipe key of the token. If the token does not have currently a wipe key, the transaction will resolve to <code>TOKENHASNOWIPEKEY</code> .
Supply	

Key	The new supply key of the token. If the token does not have currently a supply key, the transaction will resolve to <code>TOKENHASNOSUPPLYKEY</code> .
Expiration Time	The new expiry time of the token. Expiry can be updated even if the admin key is not set. If the provided expiry is earlier than the current token expiry, the transaction will resolve to <code>INVALIDEXPIRATIONTIME</code> .
Auto Renew Account	The new account which will be automatically charged to renew the token's expiration, at <code>autoRenewPeriod</code> interval.
Auto Renew Period	The new interval at which the auto-renew account will be charged to extend the token's expiry. The default auto-renew period is 7,890,000 seconds. Currently, rent is not enforced for tokens so auto-renew payments will not be made.
NOTE:	The minimum period of time is approximately 30 days (2592000 seconds) and the maximum period of time is approximately 92 days (8000001 seconds). Any other value outside of this range will return the following error: <code>AUTORENEWDURATIONNOTINRANGE</code> .
Memo	Short publicly visible memo about the token. No guarantee of uniqueness. (100 characters max)
Metadata Key	The desired new metadata key for the token. This value can be null.
Metadata	The desired new metadata for the token. This value can be null. The admin key or metadata key can be used to update this property.

```
{% hint style="info" %}
```

The sender pays the `maxAutoAssociations` fee and the rent for the first auto-renewal period for the association. This is in addition to the typical transfer fees. This ensures the receiver can receive token without association and makes it a smoother transfer process

```
{% endhint %}
```

Transaction Signing Requirements

Admin key is required to sign to update any token properties. (except for the expiry and all low priority keys)

A secondary key can update itself. Meaning it can sign a transaction that changes itself. Example: Wipe Key can sign a transaction that changes only the Wipe Key

Updating the admin key requires the new admin key to sign.

If a new treasury account is set, the new treasury key is required to sign.

The account that is paying for the transaction fee.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee.

Please use the Hedera fee estimator to estimate your transaction fee cost.

Methods

Method	Type	Requirement
<code>setTokenId(<tokenId>)</code>	TokenId	Required
<code>setTokenName(<name>)</code>	String	Optional
<code>setTokenSymbol(<symbol>)</code>	String	Optional
<code>setTreasuryAccountId(<treasury>)</code>	AccountId	Optional
<code>setAdminKey(<key>)</code>	Key	Optional
<code>setKycKey(<key>)</code>	Key	Optional
<code>setFreezeKey(<key>)</code>	Key	Optional
<code>setFeeScheduleKey(<key>)</code>	Key	


```

console.log("The transaction consensus status is " +transactionStatus);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and freeze for manual signing
tokenUpdateTransaction, err := hedera.NewTokenUpdateTransaction().
    SetTokenID(tokenId).
    SetMetadataKey(metadataKey).
    SetMetadata(newMetadata).
    SetTokenName("Your New Token Name").
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the admin private key of the token, sign with the client operator
private key and submit the transaction to a Hedera network
txResponse, err := tokenUpdateTransaction.Sign(adminKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}


```

update-nft-metadata.md:

Update NFT metadata

A TokenUpdateNftsTransaction updates the metadata property of non-fungible tokens (NFTs) on the Hedera network. The transaction requires signing with the metadata key and will fail otherwise. The new metadata must be a valid byte array and is limited to 100 bytes. All transactions are recorded on the network, providing an auditable history of changes. The metadata key allows updates to existing NFTs in a collection; if no value is provided for a field, it remains unchanged.

```
{% hint style="warning" %}
```

 Metadata keys, like other token keys, must be set during the token creation. If metadata keys are not set when the token is created, they cannot be added later, and you won't be able to update the token's metadata.

```
{% endhint %}
```

```
{% hint style="info" %}
```

With the introduction of HIP-850, the Supply Key now has the enhanced capability to update the metadata of NFTs while they are held in the treasury account. This enhancement allows for dynamic updates to NFT serial numbers before they are distributed, ensuring that once the NFTs leave the treasury account, their metadata remains immutable. This approach provides security and control, preventing unauthorized modifications after distribution.

```
{% endhint %}
```

Property	Description
Token ID	The ID of the NFT to update.
Serial Numbers	The list of serial numbers to be updated.
Metadata	The new metadata of the NFT(s).

Transaction Signing Requirements

Metadata key is required to sign.
Transaction fee payer account key.

Transaction Fees

Please see the transaction and query fees table for the base transaction fee.
Please use the Hedera fee estimator to estimate your transaction fee cost.

Methods

Method	Type	Requirement
<code>setTokenId(&#x3C;tokenId>)/code></code>	TokenID	Required
<code>setSerialNumbers(&#x3C;[int64]>)/code></code>	List<int64>	Required
<code>setMetadata(&#x3C;bytes>)/code></code>	bytes	Optional

```
{% tabs %}
{% tab title="Java" %}
{% code overflow="wrap" %}
java
// Create the transaction
TokenUpdateNftsTransaction tokenUpdateNftsTx = new TokenUpdateNftsTransaction()
    .setTokenId(tokenId)
    .setSerialNumbers(nftSerials)
    .setMetadata(newMetadata)
    .freezeWith(client);

// Sign the transaction and execute it
TransactionReceipt tokenUpdateNftsResponse =
tokenUpdateNftsTx.sign(metadataKey).execute(client);

// Get receipt for update nfts metadata transaction
TokenUpdateNftsReceipt tokenUpdateNftsReceipt =
tokenUpdateNftsResponse.getReceipt(client);

// Get the transaction consensus status
Status transactionStatus = tokenUpdateNftsReceipt.status;

// Print the token update metadata transaction status
System.out.println("Token metadata update status: " + transactionStatus);

//v2.32.0

{% endcode %}
```

```

{% endtab %}

{% tab title="JavaScript" %}
{% code overflow="wrap" %}
javascript
// Create the transaction
const tokenUpdateNftsTx = await new TokenUpdateNftsTransaction()
    .setTokenId(tokenId)
    .setSerialNumbers([nftSerials])
    .setMetadata(newMetadata)
    .freezeWith(client);

// Sign and execute the transaction with the metadata key
const tokenUpdateNftsResponse = await (
    await tokenUpdateNftsTx.sign(metadataKey)
).execute(client);

// Get receipt for token update nfts metadata transaction and log the status
const tokenUpdateNftsReceipt = await tokenUpdateNftsResponse.getReceipt(client);

// Log the token nft update transaction status
console.log(
    Token metadata update status: ${tokenUpdateNftsTxReceipt.status.toString()},
);

//v2.45.0

{% endcode %}
{% endtab %}

{% tab title="Go" %}
{% code overflow="wrap" %}
go
//Create the transaction and freeze for manual signing
tokenUpdateNftsTransaction, err := hedera.NewTokenUpdateNftsTransaction().
    SetTokenID(tokenId).
    SetSerialNumbers(nftSerials).
    SetMetadata(newMetadata).
    FreezeWith(client)
if err != nil {
    panic(err)
}

// Sign and execute the transaction
tokenUpdateNftsResponse, err :=
tokenUpdateNftsTx.Sign(metadataKey).Execute(client)
if err != nil {
    panic(err)
}

// Request the receipt of the transaction
receipt, err :=tokenUpdateNftsResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

// Print the token update metadata transaction status
fmt.Printf("Token metadata update status: ", receipt.Status)

//v2.37.0

{% endcode %}
{% endtab %}
{% endtabs %}

```


FAQs

<details>

<summary>What is the transaction fee to update a token's metadata?</summary>

The transaction fee to update the metadata of 1 NFT is \$0.001 To update metadata for multiple NFTs in a single call is $N \times \$0.001$ (N being the number of NFTs to update). See the full list of token transaction fees [here](#).

</details>

<details>

<summary>What happens if I forget to add metadata keys during token creation?</summary>

If you don't set metadata keys during token creation, you won't be able to add them later or use them to update the token's metadata.

</details>

<details>

<summary>Are metadata keys required for all token types?</summary>

No, metadata keys are not required for all token types. If your use case will need the ability to update the metadata in the future, the metadata key must be set during token creation. HIP-646 introduces the token metadata field for fungible tokens, providing users the ability to update metadata for both token types (fungible and non-fungible) using the metadata key.

</details>

<details>

<summary>Can I still create a token without metadata keys?</summary>

Yes, you can create a token without metadata keys but you won't be able to add metadata keys or update the token's metadata.

</details>

<details>

<summary>Is it possible to remove metadata keys from a token after it has been created?</summary>

No, once a token is created with metadata keys, those keys become a permanent part of the token's configuration. They cannot be removed or modified after the token creation.

</details>

<details>

<summary>Can the metadata key update the token metadata if the token is `paused` ?</summary>

No, this is just like a regular TokenUpdate. It will fail if token is paused.

</details>

<details>

<summary>Can NFT metadata be updated if the asset is held by an account that is
<code>frozen</code> for operations with that token?</summary>

If the tokenId of the NFT is not paused, and if the token has metadataKey the metadata of NFT can still be updated.

</details>

Reference: HIP-657, HIP-646, HIP-765

Contributors: MilanWR

wipe-a-token.md:

Wipe a token

Wipes the provided amount of fungible or non-fungible tokens from the specified Hedera account. This transaction does not delete tokens from the treasury account. This transaction must be signed by the token's Wipe Key. Wiping an account's tokens burns the tokens and decreases the total supply.

If the provided account is not found, the transaction will resolve to INVALIDACCOUNTID.

If the provided account has been deleted, the transaction will resolve to ACCOUNTDELETED

If the provided token is not found, the transaction will resolve to INVALIDTOKENID.

If the provided token has been deleted, the transaction will resolve to TOKENWASDELETED.

If an Association between the provided token and the account is not found, the transaction will resolve to TOKENNOTASSOCIATEDTOACCOUNT.

If Wipe Key is not present in the Token, the transaction results in TOKENHASNOWIPEKEY.

If the provided account is the token's Treasury Account, the transaction results in CANNOTWIPETOKENTREASURYACCOUNT

On success, tokens are removed from the account and the total supply of the token is decreased by the wiped amount.

The amount provided is in the lowest denomination possible.

Example: Token A has 2 decimals. In order to wipe 100 tokens from an account, one must provide an amount of 10000. In order to wipe 100.55 tokens, one must provide an amount of 10055.

This transaction accepts zero-unit token wipe operations for fungible tokens (HIP-564)

Transaction Signing Requirements:

Wipe key

Transaction fee payer account key

Transaction Fees

Please see the transaction and query fees table for the base transaction fee
Please use the Hedera fee estimator to estimate your transaction fee cost

Methods

Method	Type	Description	Requirement
<code>setTokenId(&#x3C;tokenId>)</code>		TokenId	

td><td>The ID of the fungible or non-fungible token to remove from the account.</td><td>Required</td></tr><tr><td><code>setAmount(<amount>)</code></td><td>long</td><td>Applicable to tokens of type <code>FUNGIBLECOMMON</code>.The amount of token to wipe from the specified account. The amount must be a positive non-zero number in the lowest denomination possible, not bigger than the token balance of the account.</td><td>Optional</td></tr><tr><td><code>setAccount(<accountId>)</code></td><td>AccountId</td><td>The account the specified fungible or non-fungible token should be removed from.</td><td>Required</td></tr><tr><td><code>setSerials(<serials>)</code></td><td>List<long></td><td>Applicable to tokens of type <code>NONFUNGIBLEUNIQUE</code>.The list of NFTs to wipe.</td><td>Optional</td></tr><tr><td><code>addSerial(<serial>)</code></td><td>long</td><td>Applicable to tokens of type <code>NONFUNGIBLEUNIQUE.</code>The NFT to wipe.</td><td>Optional</td></tr></tbody></table>
--

```
{% tabs %}
{% tab title="Java" %}
java
//Wipe 100 tokens from an account
TokenWipeTransaction transaction = new TokenWipeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .setAmount(100);

//Freeze the unsigned transaction, signing with the private key of the payer and
the token's wipe key; submit the transaction to a Hedera network
TransactionResponse txResponse =
transaction.freezeWith(client).sign(accountKey).sign(wipeKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Obtain the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status is " +transactionStatus);
//v2.0.1

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Wipe 100 tokens from an account and freeze the unsigned transaction for manual
signing
const transaction = await new TokenWipeTransaction()
    .setAccountId(accountId)
    .setTokenId(tokenId)
    .setAmount(100)
    .freezeWith(client);

//Sign with the payer account private key, sign with the wipe private key of the
token
const signTx = await (await transaction.sign(accountKey)).sign(wipeKey);

//Submit the transaction to a Hedera network
const txResponse = await signTx.execute(client);

//Request the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

//Obtain the transaction consensus status
const transactionStatus = receipt.status;
```

```

console.log("The transaction consensus status is "
+transactionStatus.toString());

{% endtab %}

{% tab title="Go" %}
go
//Wipe 100 tokens and freeze the unsigned transaction for manual signing
transaction, err = hedera.NewTokenBurnTransaction().
    SetAccountId(accountId).
    SetTokenID(tokenId).
    SetAmount(1000).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the payer account private key, sign with the wipe private key of the
token
txResponse, err := transaction.Sign(accountKey).Sign(wipeKey).Execute(client)

if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err = txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get the transaction consensus status
status := receipt.Status

fmt.Printf("The transaction consensus status is %v\n", status)

//v2.1.0

{% endtab %}
{% endtabs %}

```

create-an-unsigned-transaction.md:

Create an unsigned transaction

These methods allow you to build a transaction that requires further processing before it is submitted to a Hedera network. After you freeze the transaction you can use `.sign(privateKey)` to sign the transaction with multiple keys or convert the transaction to bytes for further processing.

Method	Type	Description	
<code>freeze()</code>		Freeze this transaction from further modification to prepare for signing or serialization. You will need to set the node account ID (<code>setNodeAccountId()</code>) and transaction ID (<code>setTransactionId()</code>).	
<code>freezeWith(Client client)</code>	Client	Freeze this transaction from further modification to prepare for signing or serialization. Will use the 'Client', if	

available, to generate a default Transaction ID and select 1/3 nodes to prepare this transaction

for.	<code>freezeWithSigner(<signer></code>	</td><td></td>
	Freeze the transaction with a local wallet. Local wallet available in Hedera JavaScript SDK only. >=<code>v2.11.0</code>	</td></tr></tbody></table>

```
{% tabs %}
{% tab title="Java" %}
java
//Create an unsigned transaction
AccountCreateTransaction transaction = new AccountCreateTransaction()
    .setKey(newPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000));

//Freeze the transaction for signing
//The transaction cannot be modified after this point
AccountCreateTransaction freezeTransaction = transaction.freezeWith(client);

System.out.println(freezeTransaction);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
java
//Create an unsigned transaction
const transaction = new AccountCreateTransaction()
    .setKey(newPublicKey)
    .setInitialBalance(Hbar.fromTinybars(1000));

//Freeze the transaction for signing
//The transaction cannot be modified after this point
const freezeTransaction = transaction.freezeWith(client);

console.log(freezeTransaction);

//v2.0.5

{% endtab %}

{% tab title="Go" %}
go
//Create an unsigned transaction
transaction := hedera.NewAccountCreateTransaction().
    SetKey(newKey.PublicKey()).
    SetInitialBalance(hedera.NewHbar(1000))

//Freeze the transaction for signing
//The transaction cannot be modified after this point
freezeTransaction, err := transaction.FreezeWith(client)
if err != nil {
    panic(err)
}

println(freezeTransaction.String())

//v2.0.0

{% endtab %}
{% endtabs %}
```

Sample Output

```

bash
cryptocreateaccount {
  autorenewperiod {
    seconds: 7776000
  }
  initialbalance: 1000
  key {
    ed25519: "\272g\374\310f\354\274\273bU\256\v\032$e\311\021p\216L\332\277Y\
343\230\277PUmy\373"
  }
  receiverrecordthreshold: 9223372036854775807
  sendrecordthreshold: 9223372036854775807
  nodeaccountid {
    accountnum: 6
    realmnum: 0
    shardnum: 0
  }
  transactionfee: 1000000000
  transactionid {
    accountid {
      accountnum: 9401
      realmnum: 0
      shardnum: 0
    }
    transactionvalidstart {
      nanos: 469101387
      seconds: 1604559135
    }
  }
  transactionvalidduration {
    seconds: 120
  }
}

```

get-a-transaction-receipt.md:

Get a transaction receipt

The transaction receipt gives you information about a transaction including whether or not the transaction reached consensus on the network. You request the receipt for every transaction type and there is currently no transaction fee associated with this network request.

```

{% hint style="info" %}
Receipts can be requested from a Hedera network for up to 3 minutes.
{% endhint %}

```

Transaction Receipt Contents

The transaction receipt returns the following information about a transaction:

- Whether the transaction reached consensus or not (success or fail)
- The newly generated account ID, topic ID, token ID, file ID, schedule ID, scheduled transaction ID or smart contract ID
- The exchange rate
- The topic running hash
- The topic sequence number
- The total supply of token
- The serial numbers for of the newly created NFTs after a token mint transaction was executed

Transaction Signing Requirements

Transaction receipt requests do not have an associated fee at this time so there is no signature requirement

Constructor	Description
new TransactionReceiptQuery()	Initializes the TransactionReceiptQuery Object

Transaction ID: The ID of the transaction to return the receipt for
Include Duplicates: Whether or not to include the receipts for duplicate transactions
Include Children: Whether or not to include the receipt for children transactions triggered by a parent transaction

Method	Type	Requirement
<code>setTransactionId(&#x3C;transactionId>)</code>	TransactionID	Required
<code>setIncludeDuplicates(&#x3C;value>)</code>	boolean	Optional
<code>setIncludeChildren(&#x3C;value>)</code>	boolean	Optional

```
{% tabs %}
{% tab title="Java" %}
java
new TransactionReceiptQuery()
    .setTransactionId(transactionId)
    .execute(client)
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
new TransactionReceiptQuery()
    .setTransactionId(transactionId)
    .execute(client)
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
hedera.NewTransactionReceiptQuery().
    SetTransactionId(transactionId).
    Execute(client)
```

```
{% endtab %}
{% endtabs %}
```

Helper Methods

<code>&#x3C;TransactionResponse>.getReceipt(&#x3C;client>)</code>	TransactionReceipt	Returns the receipt of a transaction
<code>&#x3C;TransactionResponse>.getReceipt(&#x3C;client, timeout>)</code>	Client, Duration	Request the receipt from the network for this duration
<code>&#x3C;TransactionResponse>.getReceiptQuery()</code>	TransactionReceiptQuery	Returns the TransactionReceiptQuery response for a transaction. This will not error on bad

```

status like RECEIPTNOTFOUND and will return information about a
failed transaction if
necessary.</td></tr><tr><td><code>#x3C;TransactionResponse>.getReceiptAsync(&#x
3C;client, timeout)</code></td><td>Client, Duration</td><td>Request receipt
asynchronously for the provided
duration</td></tr><tr><td><code>#x3C;TransactionReceipt>.status</code></
td><td>Status</td><td>Whether the transaction reached consensus or
not</td></tr><tr><td><code>#x3C;TransactionReceipt>.accountId</code></
td><td>AccountId</td><td>The newly generated account
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>.topicId</code></
td><td>TopicId</td><td>The newly generated topic
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).fileId</code></
td><td>FileId</td><td>The newly generated file
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).contractId</code></
td><td>ContractId</td><td>The newly generated contract
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).tokenId</code></
td><td>TokenId</td><td>The newly generated token
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).scheduleId</code></
td><td>ScheduleId</td><td>The newly generated schedule
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).scheduledTransactionId</
code></td><td>TransactionId</td><td>The generated scheduled transaction
ID</td></tr><tr><td><code>#x3C;TransactionReceipt>).exchangeRate</code></
td><td>ExchangeRate</td><td>The exchange rate in hbar, cents, and expiration
time</td></tr><tr><td><code>#x3C;TransactionReceipt>).topicRunningHash</code></
td><td>ByteString</td><td>The topic running
hash</td></tr><tr><td><code>#x3C;TransactionReceipt>).topicSequenceNumber</
code></td><td>long</td><td>The topic sequence
number</td></tr><tr><td><code>#x3C;TransactionReceipt>).totalSupply</code></
td><td>long</td><td>The total supply of a
token</td></tr><tr><td><code>#x3C;TransactionReceipt>).serials</code></
td><td>List#x3C;long></td><td>The list of newly created serial numbers upon
execution of a token mint transaction.</td></tr></tbody></table>

```

```

{% tabs %}
{% tab title="Java" %}
java
//Get the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

System.out.println("The transaction receipt: " +receipt);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Get the receipt of the transaction
const receipt = await txResponse.getReceipt(client);

console.log("The transaction receipt: " +receipt);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

```



```
fmt.Printf("The transaction receipt %v\n", receipt)
```

```
//v2.0.0
```

```
{% endtab %}  
{% endtabs %}
```

```
{% tabs %}  
{% tab title="Sample Output:" %}  
bash  
TransactionReceipt{  
    status=SUCCESS,  
    exchangeRate=ExchangeRate{  
        hbars=1,  
        cents=12,  
        expirationTime=2100-01-01T00:00:00Z  
    },  
    accountId=null,  
    fileId=null,  
    contractId=null,  
    topicId=null,  
    tokenId=null,  
    topicSequenceNumber=null,  
    topicRunningHash=null,  
    totalSupply=0,  
    scheduleId=0.0.2531  
    schdeduledTransactionId=null,  
    serials=[]  
}
```

```
{% endtab %}  
{% endtabs %}
```

```
# get-a-transaction-record.md:
```

```
Get a transaction record
```

You can request a transaction record for up to 3 minutes after a transaction has reached consensus. This query returns a maximum of 180 records per request. The transaction record provides the following information about a transaction:

Transaction Record Contents

Property	Description
Transaction ID	The ID of the transaction.
Consensus timestamp	The time the transaction reached consensus and was added to the ledger.
Contract Call Result	Record of the value returned by the smart contract function (if it completed and didn't fail) from ContractCallTransaction.
Contract Create Result	Record of the value returned by the smart contract constructor (if it completed and didn't fail) from ContractCreateTransaction.
Receipt	The receipt of the transaction.
Transaction Fee	The transaction fee that was charged.
Transaction Hash	The transaction hash.
Transaction Memo	The transaction memo if there was one added.
Transfers	A list of transfers made in the transaction. The list of transfers includes a payment made to the node, the service fee, and transaction fee.
Token	

Transfers	A list of the token transfers
ScheduleRef	The schedule ID of the schedule transaction the record represents.
Assessed Custom Fees	This field applies to tokens that have custom fees and returns the custom fee(s) assessed in a token transfer transaction. This includes the amount, token ID, fee collector account ID (if applicable), and effective payer account ID. The effective payer accounts are accounts that were charged the custom fees.
Automatic Associations	The token(s) that were auto associated to the account in this transaction, if any
Alias	In the record of an internal <code>AccountCreateTransaction</code> triggered by a user transaction with a (previously unused) alias, the new account's alias.
Parent Consensus Timestamp	The parent consensus timestamp is found in the record of a child transaction. The parent consensus timestamp is the consensus timestamp related to the parent transaction to this child transaction.
Ethereum Hash	The keccak256 hash of the ethereumData. This field will only be populated for EthereumTransaction.
Paid Staking Rewards	List of accounts with the corresponding staking rewards paid as a result of a transaction. See HIP-406 .
Network:	
PRNG Bytes	In the record of a PRNG transaction with no output range, a pseudorandom 384-bit string. See HIP-351 .
Network:	
PRNG Number	In the record of a PRNG transaction with an output range, the output of a PRNG whose input was a 384-bit string. See HIP-351 .
Network:	
Pending Airdrop	The pending airdrops are a result of the transaction.
Include Children	Whether or not to include the record for children transactions triggered by a parent transaction.
Include Duplicates	Whether or not to include the receipts for duplicate transactions.
Reject Airdrop	Transfer one or more tokens or token balances held by the requesting account to the treasury for each token type.

Transaction Signing Requirements

The client operator account private key is required to sign

Constructor	Description
new TransactionRecordQuery()	Initializes the TransactionRecordQuery Object

```

<table data-header-hidden>
<thead>
<tr>
<th width="434"></th>
<th width="140.33333333333331"></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
<td><strong>Type</strong></td>
<td><strong>Requirement</strong></td>
</tr>
<tr>
<td><code>setTransactionId(&#x3C;transactionId>)</code></td>
<td>TransactionId</td>
<td>Required</td>
</tr>
<tr>
<td><code>setIncludeChildren(&#x3C;value>)</code></td>
<td>boolean</td>
<td>Optional</td>
</tr>
<tr>
<td><code>setIncludeDuplicates(&#x3C;value>)</code></td>
<td>boolean</td>
<td>Optional</td>
</tr>
</tbody>
</table>

```

```

{% tabs %}
{% tab title="Java" %}
java
new TransactionRecordQuery()
    .setTransactionId(transactionId)
    .execute(client)

```

{% endtab %}

{% tab title="JavaScript" %}

```
javascript
new TransactionRecordQuery( )
    .setTransactionId(transactionId)
    .execute(client)
```

{% endtab %}

{% tab title="Go" %}

```
go
hedera.NewTransactionRecordQuery( ).
    SetTransactionId(transactionId).
    Execute(client)
```

{% endtab %}

{% endtabs %}

Methods

width="396"></th><th width="220.33333333333331"></th><th></th></tr></thead><tbody><tr><td>Method</td><td>Type</td><td>Requirement</td></tr><tr><td><code>TransactionResponse.getRecord(client)</code></td><td>TransactionRecord</td><td>Required</td></tr><tr><td><code>TransactionRecord.transactionId</code></td><td>TransactionId</td><td>Optional</td></tr><tr><td><code>TransactionRecord.consensusTimestamp</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>TransactionRecord.contractFunctionResult</code></td><td>ContractFunctionResult</td><td>Optional</td></tr><tr><td><code>TransactionRecord.receipt</code></td><td>TransactionReceipt</td><td>Optional</td></tr><tr><td><code>TransactionRecord.transactionFee</code></td><td>Hbar</td><td>Optional</td></tr><tr><td><code>TransactionRecord.transactionHash</code></td><td>ByteString</td><td>Optional</td></tr><tr><td><code>TransactionRecord.transactionMemo</code></td><td>String</td><td>Optional</td></tr><tr><td><code>TransactionRecord.transfers</code></td><td>List<Transfer></td><td>Optional</td></tr><tr><td><code>TransactionRecord.tokenTransfers</code></td><td>Map<TokenId, Map<AccountId, List<Long>>></td><td>Optional</td></tr><tr><td><code>TransactionRecord.scheduleRef</code></td><td>ScheduleId</td><td>Optional</td></tr><tr><td><code>TransactionRecord.assessedCustomFees</code></td><td>List<AssessedCustomFees></td><td>Optional</td></tr><tr><td><code>TransactionRecord.automaticTokenAssociations</code></td><td>List<TokenAssociation></td><td>Optional</td></tr><tr><td><code>TransactionRecord.ethereumHash</code></td><td>ByteString</td><td>Optional</td></tr><tr><td><code>TransactionRecord.parentConsensusTimestamp</code></td><td>Instant</td><td>Optional</td></tr><tr><td><code>TransactionRecord.paidStakingRewards</code></td><td>List<Transfer></td><td>Optional</td></tr><tr><td><code>TransactionRecord.prngBytes</code></td><td>ByteString</td><td>Optional</td></tr><tr><td><code>TransactionRecord.prngNumber</code></td><td>Integer</td><td>Optional</td></tr></tbody></table>

{% tabs %}

{% tab title="Java" %}

```

java
//Create a transaction
AccountCreateTransaction transaction = new AccountCreateTransaction()
    .setKey(newKey.getPublicKey())
    .setInitialBalance(new Hbar(1));

//Sign with the client operator account key and submit to a Hedera network
TransactionResponse txResponse = transaction.execute(client);

//Request the record of the transaction
TransactionRecord record = txResponse.getRecord(client);

System.out.println("The transaction record is " +record);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a transaction
const transaction = new AccountCreateTransaction()
    .setKey(newKey.getPublicKey())
    .setInitialBalance(new Hbar(1));

//Sign with the client operator account key and submit to a Hedera network
const txResponse = await transaction.execute(client);

//Request the record of the transaction
const record = await txResponse.getRecord(client);

console.log("The transaction record is " +record);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create a transaction
transaction := hedera.NewAccountCreateTransaction().
    SetKey(privateKey.PublicKey()).
    SetInitialBalance(hedera.NewHbar(1000))

//Sign with the client operator account key and submit to a Hedera network
txResponse, err := transaction.Execute(client)

//Request the record of the transaction
record, err := txResponse.GetRecord(client)

fmt.Printf("The transaction record is %v\n", record)

//v2.0.0

{% endtab %}
{% endtabs %}

{% tabs %}
{% tab title="Sample Output:" %}

TransactionRecord{
    receipt=TransactionReceipt{
        status=SUCCESS,
        exchangeRate=ExchangeRate{

```

```

        hbars=30000, cents=116646,
        expirationTime=2020-09-04T03:00:00Z
    },
    accountId=0.0.97001,
    fileId=null,
    contractId=null,
    topicId=null,
    topicSequenceNumber=null
},
transactionHash=e005670a1f49c4fd776b2d432db3e5cb31441 bb5a35bfff412ec3b41cb1
3366ce00b5c1b9900aad1467f9709a649ccc20,
consensusTimestamp=2020-11-05T08:34:31.107311002Z,
transactionId=0.0.9401@160456525 8.479476328,
transactionMemo=,
transactionFee=0.25401241 ¢,
contractFunctionResult=null,
transfers=[
    Transfer{accountId=0.0.5, amount=0.01501152 ¢}, (node fee)
    Transfer{accountId=0.0.98, amount=0.23900089 ¢}, (service fee)
    Transfer{accountId=0.0.9401, amount=-1.25401241 ¢}, (transaction fee)
]
initial balance of new account
    Transfer{accountId=0.0.97001, amount=1 ¢} (Initial balance of the new
account)
]
tokenTransfers={},
scheduleRef=null,
assessedCustomFees=[],
automaticTokenAssociations=[TokenAssociation{
    tokenId=0.0.27335, accountId=0.0.27333}]
}

{% endtab %}
{% endtabs %}

```

manually-sign-a-transaction.md:

Manually sign a transaction

Sign a transaction using the private key(s) required to sign the transaction. You cannot sign the transaction with a public key. If your client operator account private key is the key used in the key field(s) of a transaction, you do not need to manually sign the transaction. The `execute(client)` method signs the transaction with the client operator account private key before it is submitted to a Hedera network.

Method	Type	Description
<code>sign(privateKey)</code>	PrivateKey	Sign the transaction with an ED25519 private key
<code>signWith(publicKey, transactionSigner)</code>	PublicKey, TransactionSigner	Sign the transaction with a callback that may block waiting for user confirmation.
<code>signWithOperator(client)</code>	Client	Sign the transaction with the client
<code>signWithSigner(signer)</code>		Sign the transaction with a local wallet. Local wallet available in Hedera JavaScript SDK only. >=v2.11.0

```

{% tabs %}
{% tab title="Java" %}
java
//Create any transaction

```

```

AccountUpdateTransaction transaction = new AccountUpdateTransaction()
    .setAccountId(accountId)
    .setKey(key);

//Freeze the transaction for signing
AccountUpdateTransaction freezeTransaction = transaction.freezeWith(client);

//Sign the transaction with a private key
AccountCreateTransaction signedTransaction = freezeTransaction
    .sign(PrivateKey.fromString("302e020100300506032b65700422042012a4a4add3d885b
d61d7ce5cff88c5ef2d510651add00a7f64cb90de3359bc5c"));

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create any transaction
const transaction = await new AccountUpdateTransaction()
    .setAccountId(accountId)
    .setKey(key)
    .freezeWith(client);

//Sign the transaction with a private key
const signedTransaction = transaction
    .sign(PrivateKey.fromString("302e020100300506032b65700422042012a4a4add3d885b
d61d7ce5cff88c5ef2d510651add00a7f64cb90de3359bc5c"));

{% endtab %}

{% tab title="Go" %}
go
//Create any transaction
transaction := hedera.NewAccountUpdateTransaction().
    SetAccountId(newAccountId).
    SetKey(updateKey.PublicKey())

//Freeze the transaction for signing
freezeTransaction, err := transaction.FreezeWith(client)
if err != nil {
    panic(err)
}

signedTransaction :=
freezeTransaction.Sign(hedera.PrivateKeyFromString("302e020100300506032b65700422
042012a4a4add3d885bd61d7ce5cff88c5ef2d510651add00a7f64cb90de3359bc5c"))
//v2.0.0

{% endtab %}
{% endtabs %}

```

modify-transaction-fields.md:

Modify transaction fields

When submitting a transaction to the Hedera network, various fields can be modified, such as the transaction ID, consensus time, memo field, account ID of the node, and the maximum fee. These values can be set using methods provided by the SDKs. However, they are not required as the SDK can automatically create or use default values.

```
{% hint style="info" %}
```

Note: The total size for a given transaction is limited to 6KiB.

{% endhint %}

Fields	Description
-----	-----
Transaction ID	<p>Set the ID for this transaction. The transaction ID includes the operator's account (the account paying the transaction fee). If two transactions have the same transaction ID, they won't both have an effect. One will complete normally and the other will fail with a duplicate transaction status.</p> <p>Normally, you should not use this method. Just before a transaction is executed, a transaction ID will be generated from the operator on the client.</p>
Valid Duration	<p>Set the duration that this transaction is valid for</p> <p>Note: Max network valid duration is 180 seconds. SDK default value is 120 seconds. The minium valid duration period is 15 seconds.</p>
Memo	<p>Set a note or description that should be recorded in the transaction record (maximum length of 100 characters). Anyone can view this memo on the network</p>
Node ID	<p>Set the account ID of the node that this transaction will be submitted to.</p>
Max transaction fee	<p>Set the max transaction fee for the operator (transaction fee payer account) is willing to pay</p> <p>Default: 2 hbar</p>

{% hint style="info" %}

Note: The SDKs do not require you to set these fields when submitting a transaction to a Hedera network. All methods below are optional and can be used to modify any fields.

{% endhint %}

Method	Type
<code>setTransactionID(&#x3C;transactionId>)</code>	TransactionID
<code>setTransactionValidDuration(&#x3C;validDuration>)</code>	Duration
<code>setTransactionMemo(&#x3C;memo>)</code>	String
<code>setNodeAccountIds(&#x3C;nodeAccountIds>)</code>	List<#x3C;AccountId>
<code>setMaxTransactionFee(&#x3C;maxTransactionFee>)</code>	Hbar
<code>setGrpcDeadline(&#x3C;grpcDeadline>)</code>	Duration
<code>setRegenerateTransactionId(&#x3C;regenerateTransactionId>)</code>	boolean

{% tabs %}

{% tab title="Java" %}

java

//Create the transaction and set the transaction properties

Transaction transaction = new AccountCreateTransaction() //Any transaction can be applied here

.setKey(key)

.setInitialBalance(Hbar.fromTinybars(1000))

.setMaxTransactionFee(new Hbar(2)) //Set the max transaction fee to 2 hbar

.setTransactionMemo("Transaction memo"); //Set the node ID to submit the transaction to

```
//v2.0.0
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Create the transaction and set the transaction properties
```

```
const transaction = await new AccountCreateTransaction() //Any transaction can be applied here
```

```
    .setKey(key)
```

```
    .setInitialBalance(Hbar.fromTinybars(1000))
```

```
    .setMaxTransactionFee(new Hbar(2)) //Set the max transaction fee to 2 hbar
```

```
    .setTransactionMemo("Transaction memo"); //Set the node ID to submit the transaction to
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```
go
```

```
//Create the transaction and set the transaction properties
```

```
transaction := hedera.NewAccountCreateTransaction(). //Any transaction can be applied here
```

```
    SetKey(newKey.PublicKey()).
```

```
    SetInitialBalance(hedera.NewHbar(1000)).
```

```
    SetMaxTransactionFee(hedera.NewHbar(2)). //Set the max transaction fee to 2 hbar
```

```
    SetTransactionMemo("Transaction memo") //Set the transaction memo
```

```
//v2.0.0
```

```
{% endtab %}
```

```
{% endtabs %}
```

Get transaction properties

```
<table><thead><tr><th
width="438">Method</th><th>Type</th></tr></thead><tbody><tr><td><code>getTransac
tionID()</code></td><td>TransactionID</td></
tr><tr><td><code>getTransactionValidDuration()</code></td><td>Duration</td></
tr><tr><td><code>getTransactionMemo()</code></td><td>String</td></
tr><tr><td><code>getNodeAccountId()</code></td><td>AccountId</td></
tr><tr><td><code>getMaxTransactionFee()</code></td><td>Hbar</td></
tr><tr><td><code>getTransactionHash()</code></td><td>byte[
]</td></tr><tr><td><code>getTransactionHashPerNode()</code></td><td>Map<#x3C;Acc
ountId, byte [
]></td></tr><tr><td><code>getSignatures()</code></td><td>Map<#x3C;AccountId,
Map<#x3C;PublicKey, byte [
]>></td></tr></tbody></table>
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Create the transaction and set the transaction properties
```

```
Transaction transaction = new AccountCreateTransaction() //Any transaction can be applied here
```

```
    .setInitialBalance(Hbar.fromTinybars(1000))
```

```
    .setMaxTransactionFee(new Hbar(50)) //Set the max transaction fee to 50 hbar
```

```
    .setNodeAccountId(new AccountId(3)) //Set the node ID to submit the transaction to
```

```
    .setTransactionMemo("Transaction memo"); //Add a transaction memo
```

```
Hbar maxTransactionFee = transaction.getMaxTransactionFee();
```

```
//v2.0.0
```



```
{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction and set the transaction properties
const transaction = await new AccountCreateTransaction() //Any transaction can
be applied here
    .setInitialBalance(Hbar.fromTinybars(1000))
    .setMaxTransactionFee(new Hbar(50)) //Set the max transaction fee to 50 hbar
    .setNodeAccountId(new AccountId(3)) //Set the node ID to submit the
transaction to
    .setTransactionMemo("Transaction memo"); //Add a transaction memo

const maxTransactionFee = transaction.getMaxTransactionFee();

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction and set the transaction properties
transaction := hedera.NewAccountCreateTransaction(). //Any transaction can be
applied here
    SetKey(newKey.PublicKey()).
    SetInitialBalance(hedera.NewHbar(100)).
    SetMaxTransactionFee(hedera.NewHbar(2)). //Set the max transaction fee to 2
hbar
    SetTransactionMemo("Transaction memo") //Add a transaction memo

maxtransactionFee := transaction.GetMaxTransactionFee()
//v2.0.0

{% endtab %}
{% endtabs %}
```

README.md:

Transactions

Transactions are requests submitted by a client to a node in the Hedera network. Every transaction has a fee that will be paid for processing the transaction. The following table lists the transaction type requests for each service.

```
{% hint style="info" %}
Transactions have a 6,144-byte size limit. This includes the signatures on the
transaction. The estimated single signature size is about 80-100 bytes.
{% endhint %}
```

```
{% hint style="info" %}
```

Transfers\

With the R4 release, how the HBAR balance changes of accounts involved in the transaction (either directly or node) are represented within the transaction record has been modified. In the past, each transfer of HBAR, whether a payment from one account to another or a fee paid to a Hedera node or to Hedera – was listed individually. The list of transfers might include one for the payer making the fundamental payment, one for that same account paying a fee to the network, and another for the same account paying a fee to the node. The new model combines all those individual transfers and shows, for each account involved in the transaction, only the net transfer value.

```
{% endhint %}

<table data-full-width="true"><thead><tr><th width="243">Cryptocurrency
Accounts</th><th width="203">Consensus</th><th>Tokens</th><th>File
Service</th><th>Smart Contracts</th></tr></thead><tbody><tr><td><a
```

```

href="../accounts-and-hbar/create-an-account.md">AccountCreateTransaction</a></td><td><a href="../consensus-service/create-a-topic.md">TopicCreateTransaction</a></td><td><a href="../token-service/define-a-token.md">TokenCreateTransaction</a></td><td><a href="../file-service/create-a-file.md">FileCreateTransaction</a></td><td><a href="../smart-contracts/create-a-smart-contract.md">ContractCreateTransaction</a></td></tr><tr><td><a href="../accounts-and-hbar/update-an-account.md">AccountUpdateTransaction</a></td><td><a href="../consensus-service/update-a-topic.md">TopicUpdateTransaction</a></td><td><a href="../token-service/update-a-token.md">TokenUpdateTransaction</a></td><td><a href="../file-service/append-to-a-file.md">FileAppendTransaction</a></td><td><a href="../smart-contracts/update-a-smart-contract.md">ContractUpdateTransaction</a></td></tr><tr><td><a href="../accounts-and-hbar/transfer-cryptocurrency.md">TransferTransaction</a></td><td><a href="../consensus-service/submit-a-message.md">TopicMessageSubmitTransaction</a></td><td><a href="../token-service/delete-a-token.md">TokenDeleteTransaction</a></td><td><a href="../file-service/update-a-file.md">FileUpdateTransaction</a></td><td><a href="../smart-contracts/delete-a-smart-contract.md">ContractDeleteTransaction</a></td></tr><tr><td><a href="../accounts-and-hbar/delete-an-account.md">AccountDeleteTransaction</a></td><td><a href="../consensus-service/delete-a-topic.md">TopicDeleteTransaction</a></td><td><a href="../token-service/associate-tokens-to-an-account.md">TokenAssociateTransaction</a></td><td><a href="../file-service/delete-a-file.md">FileDeleteTransaction</a></td><td><a href="../smart-contracts/ethereum-transaction.md">EthereumTransaction</a></td></tr><tr><td><a href="../accounts-and-hbar/approve-an-allowance.md">AccountAllowanceApprovalTransaction</a></td><td></td><td><a href="../token-service/dissociate-tokens-from-an-account.md">TokenDissociateTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/mint-a-token.md">TokenMintTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/burn-a-token.md">TokenBurnTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/freeze-an-account.md">TokenFreezeTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/update-a-fee-schedule.md">TokenFeeScheduleUpdateTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/unfreeze-an-account.md">TokenUnfreezeTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/enable-kyc-account-flag.md">TokenGrantKycTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/disable-kyc-account-flag.md">TokenRevokeKycTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/pause-a-token.md">TokenPauseTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/unpause-a-token.md">TokenUnpauseTransaction</a></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td><a href="../token-service/wipe-a-token.md">TokenWipeTransaction</a></td><td></td><td></td></tr></tbody></table>

```

sign-a-multisignature-transaction.md:

Sign a multisignature transaction

Hedera supports multisignature transactions. This means a Hedera transaction can require more than one key to sign a transaction in order for it to be processed on a Hedera network. These keys can be set up as a key list where all the keys in the specified list are required to sign the transaction or a threshold key where only a subset of the keys from a specified list are required to sign the transaction. The example below shows how you can use multiple keys to sign and submit a transaction.

```
{% hint style="info" %}
```

Note: This example uses version 2.0 of the SDKs.
{% endhint %}

1. Create the transaction

In this example, we will use a transfer transaction that requires 3 keys to sign the transaction. If all 3 keys do not sign the transaction, the transaction will not execute and an "INVALIDSIGNATURE" response will be returned from the network. The senderAccountId is a Hedera account that was created with a key list of 3 keys. Since the sender account is required to sign in a transfer transaction, all three keys are required to sign to complete the transfer of hbars from the sender account to the recipient account. The recipient account is not required to sign the transaction.

After you create the transfer transaction, you will need to freeze (freezeWith(client)) the transaction from further modification so that transaction cannot be tampered with. This ensures each signer is signing the same exact transaction. The transaction is then shared with each of the three signers to sign with their private keys.

{% hint style="info" %}

It is required to set the account ID of the node(s) the transaction will be submitted to when freezing a transaction for signatures. To set the node account ID(s) you apply the .setNodeAccountIds() method.

{% endhint %}

{% tabs %}

{% tab title="Java" %}

java

//The node account ID to submit the transaction to. You can add more than 1 node account ID to the list.

```
List<AccountId> nodeId = Collections.singletonList(new AccountId(3));
```

//Create the transfer transaction

```
TransferTransaction transferTransaction = new TransferTransaction()  
    .addHbarTransfer(senderAccountId, new Hbar(1))  
    .addHbarTransfer(receiverAccountId, new Hbar(-1))  
    .setNodeAccountIds(nodeId);
```

//Freeze the transaction from any further modifications

```
TransferTransaction transaction = transferTransaction.freezeWith(client);
```

{% endtab %}

{% tab title="JavaScript" %}

javascript

//The node account ID to submit the transaction to. You can add more than 1 node account ID to the list

```
const nodeId = [];
```

```
nodeId.push(new AccountId(3));
```

//Create the transfer transaction

```
const transferTransaction = new TransferTransaction()  
    .addHbarTransfer(senderAccountId, new Hbar(1))  
    .addHbarTransfer(receiverAccountId, new Hbar(-1))  
    .setNodeAccountIds(nodeId);
```

//Freeze the transaction from further modifications

```
const transaction = await transferTransaction.freezeWith(client);
```

{% endtab %}

{% tab title="Go" %}

go

```

nodeAccountId, err := hedera.AccountIDFromString("0.0.3")
if err != nil {
}

nodeIdList := []hedera.AccountID{nodeAccountId}

//Create the transfer transaction
transferTransaction := hedera.NewTransferTransaction().
    AddHbarTransfer(senderAccountId, hedera.NewHbar(1)).
    AddHbarTransfer(receiverAccountId, hedera.NewHbar(-1)).
    SetNodeAccountIDs(nodeIdList)

//Freeze the transaction from any further modifications
transaction, err := transferTransaction.FreezeWith(client)

{% endtab %}
{% endtabs %}

```

2. Collect required signatures

Each signer can sign the transaction with their private key. The signed transactions are then returned to you from each of the 3 signers, resulting in 3 separate signature bytes. The sample code below is for illustrative purposes as you would not have the private keys to sign the transaction for each signer.

```

{% tabs %}
{% tab title="Java" %}
java
//Signer one signs the transaction with their private key
byte[] signature1 = privateKeySigner1.signTransaction(transaction);

//Signer two signs the transaction with their private key
byte[] signature2 = privateKeySigner2.signTransaction(transaction);

//Signer three signs the transaction with their private key
byte[] signature3 = privateKeySigner3.signTransaction(transaction);

//signature1, signature2, and signature3 are returned back to you

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Signer one signs the transaction with their private key
const signature1 = privateKeySigner1.signTransaction(transaction);

//Signer two signs the transaction with their private key
const signature2 = privateKeySigner2.signTransaction(transaction);

//Signer three signs the transaction with their private key
const signature3 = privateKeySigner3.signTransaction(transaction);

//signature1, signature2, and signature3 are returned back to you

{% endtab %}

{% tab title="Go" %}
go
//Signer one signs the transaction with their private key
signature1, err := privateKeySigner1.SignTransaction(transaction.Transaction)
if err != nil {
}

//Signer two signs the transaction with their private key

```

```
signature2, err := privateKeySigner2.SignTransaction(transaction.Transaction)
if err != nil {
}
```

```
//Signer three signs the transaction with their private key
signature3, err := privateKeySigner3.SignTransaction(transaction.Transaction)
if err != nil {
}
```

//signature1, signature2, and signature3 are returned back to you

```
{% endtab %}
{% endtabs %}
```

3. Create a single transaction with all three signatures

Once you have collected all three signature bytes (signature1, signature2, signature 3), you will then apply them to the transaction (transaction) to create a single transaction with all three signatures to submit to the network. You will take the transaction that we started with and apply the signature bytes to the transaction using addSignature(). You will need the public keys of each of the signers to include in the method.

You can add as many signatures to a single transaction as long as the transaction size does not exceed 6,144 kb. Each additional signature applied to the transaction will increase the transaction fee from its base cost of \$0.0001. If the transaction fee increases beyond the SDK's default max transaction fee of 1 hbar, you will need to update the max transaction fee value by calling the .setMaxTransactionFee() before submitting the transaction to the network.

```
{% tabs %}
{% tab title="Java" %}
java
//Collate all three signatures with the transaction
TransferTransaction signedTransaction = transaction.addSignature(publicKey1,
signature1).addSignature(publicKey2, signature2).addSignature(publicKey3,
signature3);
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
javascript
//Collate all three signatures with the transaction
const signedTransaction = transaction.addSignature(publicKey1,
signature1).addSignature(publicKey2, signature2).addSignature(publicKey3,
signature3);
```

```
{% endtab %}
```

```
{% tab title="Go" %}
go
//Collate all three signatures with the transaction
signedTransaction := transaction.AddSignature(publicKey1,
signature1).AddSignature(publicKey2, signature2).AddSignature(publicKey3,
signature3)
```

```
{% endtab %}
{% endtabs %}
```

4. Verify the required signers public keys

Before you submit the transaction to a Hedera network, you may want to verify the keys that signed the transaction. You can view the public keys that signed the transaction by calling .getSignatures() to signedTransaction. You can then

compare the public keys returned from the `signedTransaction.getSignatures()` to the public keys of each of the signers to verify the required keys have signed. Both the public key and public key bytes are returned.

```
{% tabs %}
{% tab title="Java" %}
java
//Print all public keys that signed the transaction
System.out.println("The public keys that signed the transaction "
+signedTransaction.getSignatures());

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Print all public keys that signed the transaction
console.log("The public keys that signed the transaction "
+signedTransaction.getSignatures());

{% endtab %}

{% tab title="Go" %}
go
//Print all public keys that signed the transaction
signatures, err := signedTransaction.GetSignatures()
fmt.Println("The public keys that signed the transaction ", signatures)

{% endtab %}
{% endtabs %}
```

5. Submit the transaction

We are now ready to submit the transfer transaction to a Hedera network. To submit the transaction, we will apply the `.execute()` method to `signedTransaction`. After the transaction is submitted, we will print the transaction ID to the console. You can use the transaction ID to search for transaction details in a mirror node explorer like DragonGlass, Hashscan, or Ledger Works. Be sure to select the correct network when searching for the transaction in a mirror node explorer. You can also check the status of a transaction by requesting the receipt of the transaction and obtaining the status.

```
{% tabs %}
{% tab title="Java" %}
java
//Submit the transaction to a Hedera network
TransactionResponse submitTx = signedTransaction.execute(client);

//Get the transaction ID
TransactionId txId = submitTx.transactionId;

//Print the transaction ID to the console
System.out.println("The transaction ID " +txId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Submit the transaction to a Hedera network
const submitTx = await signedTransaction.execute(client);

//Get the transaction ID
const txId = submitTx.transactionId.toString();
```

```
//Print the transaction ID to the console
console.log("The transaction ID " +txId);

{% endtab %}

{% tab title="Go" %}
go
//Submit the transaction to a Hedera network
submitTx, err := signedTransaction.Execute(client)

//Get the transaction ID
txId := submitTx.TransactionID

//Print the transaction ID to the console
fmt.Println("The transaction ID ", txId)

{% endtab %}
{% endtabs %}
```

submit-a-transaction.md:

Submit a transaction

The execute() method submits a transaction to a Hedera network. This method will create the transaction ID from the client operator account ID, sign with the client operator private key, and pick a node from the defined network on the client to submit the transaction to. The transaction is also automatically signed with the client operator account private key. You do not need to manually sign transactions if this key is the required key on any given transaction. Once you submit the transaction, the response will include the following:

- The transaction ID of the transaction
- The node ID of the node the transaction was submitted to
- The transaction hash

Transaction Signing Requirements

Please refer to the specific transaction type and defined key structure of the account, topic, token, file, or smart contract to understand the signing requirements

Method	Type	Description
<code>execute(client)</code>	Client	Sign with the client operator and submit to a Hedera network
<code>execute(client, timeout)</code>	Client, Duration	The duration of times the client will try to submit the transaction upon the network being busy
<code>executeWithSigner(signer)</code>		Sign the transaction with a local wallet. This feature is available in the Hedera JavaScript SDK only.
<code>v2.11.0</code>		
<code><transactionResponse>.transactionId</code>	TransactionId	Returns the transaction ID of the transaction
<code><transactionResponse>.nodeId</code>	AccountId	Returns the node ID of the node that processed the transaction
<code><transactionResponse>.transactionHash</code>	byte []	Returns the hash of the transaction
<code><transactionResponse>.setValidateStatus(validateStatus)</code>	boolean	Whether getReceipt() or getRecord() will throw an exception if the receipt status is not SUCCESS
<code><transactionResponse>.getValidateStatus</code>	boolean	Return whether getReceipt() or getRecord() will throw an exception if the receipt status is not

```
SUCCESS</td></tr></tbody></table>
```

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction
AccountCreateTransaction transaction = new AccountCreateTransaction()
    .setKey(newPublicKey)
    .setInitialBalance(new Hbar(5));

//Sign with client operator private key and submit the transaction to a Hedera
network
TransactionResponse txResponse = transaction.execute(client);

//Get the transaction ID
TransactionId transactionId = txResponse.transactionId;

//Get the account ID of the node that processed the transaction
AccountId nodeId = txResponse.nodeId;

//Get the transaction hash
byte [] transactionHash = txResponse.transactionHash;

System.out.println("The transaction ID is " +transactionId);
System.out.println("The transaction hash is " +transactionHash);
System.out.println("The node ID is " +nodeId);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction
const transaction = new AccountCreateTransaction()
    .setKey(newPublicKey)
    .setInitialBalance(new Hbar(5));

//Sign with client operator private key and submit the transaction to a Hedera
network
const txResponse = await transaction.execute(client);

//Get the transaction ID
const transactionId = txResponse.transactionId;

//Get the account ID of the node that processed the transaction
const nodeId = txResponse.nodeId;

//Get the transaction hash
const transactionHash = txResponse.transactionHash;

console.log("The transaction ID is " +transactionId);
console.log("The transaction hash is " +transactionHash);
console.log("The node ID is " +nodeId);

//v2.0.0

{% endtab %}

{% tab title="Go" %}
java
//Create the transaction
transaction := hedera.NewAccountCreateTransaction().
    SetKey(publicKey).
```



```

        SetInitialBalance(hedera.NewHbar(5))

//Sign with client operator private key and submit the transaction to a Hedera
network
txResponse, err := transaction.Execute(client)

if err != nil {
    panic(err)
}

//Get the transaction ID
transactionId := txResponse.TransactionID

//Get the account ID of the node that processed the transaction
transactionNodeId := txResponse.NodeID

//Get the transaction hash
transactionHash := txResponse.Hash

fmt.Printf("The transaction id is %v\n", transactionId)
fmt.Printf("The transaction hash is %v\n", transactionHash)
fmt.Printf("The node id is %v\n", transactionNodeId)

//v2.0.0

{% endtab %}
{% endtabs %}

```

transaction-id.md:

Transaction ID

A transaction ID is composed of the payer account ID and the timestamp in seconds.nanoseconds format (0.0.9401@1602138343.335616988). You are not required to generate a transaction ID for every transaction type as the SDKs generate them when submitting transactions.

Schedule Transaction ID

Scheduled transactions have a schedule flag in the transaction ID (0.0.9401@1602138343.335616988?schedule).

Child Transaction ID

Child transactions are transactions that were triggered by a parent transaction. Child transactions have a nonce populated in the transaction ID after the timestamp. The nonce value for the parent transaction ID is 0. The transaction ID (payer and timestamp) is the same as the parent transaction for each child transaction. Each child transaction adds a nonce value to the parent transaction ID. For example, a parent transaction with one child transaction would result in the child transaction having a nonce value of 1 (0.0.2252@1640075693.891386528/1). The parent transaction ID for the child transaction would be 0.0.2252@1640075693.891386528.

Constructor	Description
new TransactionId()	Initializes the TransactionId object

Generate a transaction ID

Method	Type	Description

<code>setNonce(&#x3C;nonce>)/code></code>	Integer	Set the nonce for the child transaction
<code>setScheduled(&#x3C;schedule>)/code></code>	boolean	Set the boolean value for a scheduled transaction
<code>TransactionId.generate(&#x3C;accountId>)/code></code>	AccountId	Generates a new transaction ID. Pass the payer account ID to generate the transaction
<code>TransactionId.fromBytes(&#x3C;bytes>)/code></code>	byte []	Converts to a transaction ID from bytes
<code>TransactionId.fromString(&#x3C;string>)/code></code>	String	Converts a string to transaction ID
<code>TransactionId.withValidStart(&#x3C;accountId>, &#x3C;validStart>)/code></code>	AccountId, Instant	Create a transaction ID by passing the payer account and valid start time

```
{% tabs %}
{% tab title="Java" %}
java
TransactionId txId = TransactionId.generate(new AccountId(5));
System.out.println(txId);

//v2.0.0

{% endtab %}

{% tab title="JavaScript" %}
javascript
const txId = TransactionId.generate(new AccountId(5));
console.log(txId);
//v2.0.0

{% endtab %}

{% tab title="Go" %}
go
txId := hedera.TransactionIDGenerate(client.GetOperatorAccountID())
fmt.Println(txId)

//v2.0.0

{% endtab %}
{% endtabs %}
```

Sample Output:

```
0.0.5@1604557331.565419523
```

brand-guidelines.md:

Brand Guidelines

Incorporate official Hedera logos and iconography

The Decentralized on Hedera stamp is the official way to denote that your application uses the Hedera Hashgraph network. We think by using it it demonstrates to your users the trust that Hedera can provide to them.

For use on white or light backgrounds.

```

```

For use on black or dark backgrounds

For Hedera logo usage, details, and more view the complete Hedera Brand page.

contributing-guide.md:

```
---
description: >-
  Learn how to submit a demo application, create pull requests, or log issues in
  the Hedera Contributing Guide.
cover: broken-reference
coverY: 31
---
```

Contributing & Style Guide

We value every form of contribution, no matter how small. In this guide, you will find steps on submitting an issue, creating a pull request, submitting a demo application, creating Hedera Improvement Proposals (HIPs), and adhering to our style guide. Thanks in advance for your contributions!

1. CREATE ISSUES	#submit-an-issue
2. PULL REQUESTS	#creating-a-pull-request
3. HIPs	#hedera-improvement-proposal-hip
4. DEMO APPLICATIONS	#submit-demo-applications
5. STYLE GUIDE	#style-guide
📄 REPO	https://github.com/hashgraph/hedera-docs

Create Issues

If you've identified a problem in the documentation or have a suggestion for additional content, you can submit an issue. Follow these steps:

1. Navigate to the Repository: Visit the hedera-docs repository on GitHub.
2. Open the Issues Tab: Find the "Issues" tab near the top of the repository page, next to "Code" and "Pull Requests". Click on it.
3. Create a New Issue: Click the "New Issue" button to open a form where you can detail the problem or suggestion.
4. Fill Out the Form: Give your issue a title and a detailed description. Be as clear and concise as possible to ensure we fully understand the issue. If applicable, include screenshots or code snippets.
5. Submit the Issue: After filling out the issue, click the "Submit new issue" button. We'll review your issue and take appropriate action.

Create Pull Requests

If you'd like to propose changes directly to the documentation, you can submit a pull request. Here's how:

1. Fork the Repository: Navigate to the hedera-docs repository and click the "Fork" button at the top right. This creates a copy of the repository in your GitHub account.
2. Clone the Forked Repository: Clone the forked repository to your local system and make changes. Be sure to follow the repository's coding and style guidelines.
3. Commit Your Changes: Once you've made your changes, commit them with a clear, detailed message describing the changes you've made.
 - 1. Use sign-off when making each of your commits.
 - 1. Alternatively, you can use auto sign-off by installing `cp hooks-git/prepare-commit-msg .git/hooks && chmod +x .git/hooks/prepare-commit-msg`.
 - 2. Use this guide to install the pre-commit hook scripts to check for files with names that would conflict on a case-insensitive filesystem like MacOS HFS+ or Windows FAT.
4. Push Your Changes: Push your committed changes to your forked repository on GitHub.
5. Submit a Pull Request: Back in the hedera-docs repository, click the "Pull Requests" tab and then the "New pull request" button. Select your forked repository and the branch containing your changes, then click "Create pull request".
6. Describe Your Changes: Give your pull request a title and describe the proposed changes. This description should make it clear why the changes should be incorporated.
7. Submit the Pull Request: Click the "Create pull request" button to submit it. We'll review your proposed changes and, if they're approved, merge them into the repository.

By logging issues and creating pull requests, you're helping us make the Hedera documentation better for everyone. We appreciate your contributions and look forward to collaborating with you!

{% hint style="info" %}

Note: The Hedera team will review issues and pull requests.

{% endhint %}

Hedera Improvement Proposal (HIP)

Have a new feature request for consensus or mirror nodes? Looking to submit a standard or informational guide for the Hedera ecosystem? Submit a Hedera Improvement Proposal that will be reviewed and evaluated by the Hedera Team. These improvement proposals can range from core protocol changes to the applications, frameworks, and protocols built on the Hedera public network and used by the community. To view all active and pending HIPs, check out the HIP website.

1. Fork the [hedera-improvement-proposal](#) repository here.
2. Fill out this HIP template.
3. Create a pull request against [hashgraph/hedera-improvement-proposal](#) main branch.

{% hint style="info" %}

Note: Which category should you make the HIPs? See [hip-1](#) for details on the HIP process, or watch the following video tutorial.

{% endhint %}

{% embed url="https://www.youtube.com/watch?v=Gbk8EbtibA0" %}

Hedera Improvement Proposals\

by Developer Advocate: Michael Garber

{% endembed %}

Submit Demo Applications

If you have a demo application that you'd like to share, we encourage you to follow the steps outlined below to ensure your application is showcased accurately.

1. Open an Enhancement issue in the hedera-docs repository.
2. Within the issue, please include the following details:
 1. Demo application name: The official name of your demo application. This is how it will be listed on the demo applications page.
 2. Developer/Maintainer name and GitHub username: Your name or the person maintaining the demo application. This ensures we know who to contact for any future updates or questions regarding the application.
 3. Link to the demo application GitHub repository: Please provide a link to the public GitHub repository where your demo application is hosted. This allows the Hedera community to access and review your application.
3. Submit your issue once you've provided the required details within the issue. Our team will review your submission, and if approved, your demo application will be added to our list.

Remember, the aim is to showcase applications that demonstrate the potential and functionality of Hedera in various use cases. Clear, concise, and well-documented code is highly appreciated.

Thank you for your valuable contribution to the Hedera community! We look forward to reviewing your demo application.

Style Guide

<details>

<summary>GitBook Markdown Syntax</summary>

Please refer to the GitBook Markdown Syntax guide.

</details>

<details>

<summary>Use of HBAR</summary>

When referring to the Hedera native currency, use the singular form of the noun HBAR. For example:

"I bought 10 HBAR yesterday"

Do not use the plural form of the noun, as this style rule applies even when referring to multiple units of HBAR.

tinybars

When referring to fractions of HBAR, use the plural form tinybars. For example:

"I will transfer 1,000 tinybars from my account to yours"

Do not use the singular form of the noun, as any reference should be plural since one HBAR equals 100,000 tinybars.

</details>

<details>

<summary>Use of web2 and web3</summary>

When documenting or referring to "web2" and "web3," it's important to maintain consistency. Both terms should be in lowercase. The only exception to this rule is when either term starts a sentence. In such cases, the initial letter should be capitalized. For example:

- ✗ Incorrect: "web3 technologies are evolving rapidly."
- ✓ Correct: "Web3 represents a shift towards decentralization."
- ✓ Correct: "In the context of web2, user data is often controlled by centralized entities."
- ✓ Correct: "The principles of transparency and user empowerment are fundamental to the development of web3 platforms."

</details>

<details>

<summary>American English</summary>

Follow the American English spelling standard. This means that words should follow the American English conventions, employing 'z' instead of 's' in words such as 'decentralized,' 'realized,' and 'organized.'

For example:

- Use 'color' instead of the British English 'colour.'
- Use 'analyze' instead of the British English 'analyse.'
- Use 'organization' instead of the British English 'organisation.'

Use an American English dictionary or a recognized American English style guide to ensure consistency and accuracy throughout the text. Tools like Grammarly or spell checkers set to American English can assist in maintaining this standard.

</details>

<details>

<summary>Tutorial steps</summary>

When presenting steps or instructions within the documentation, the following guidelines should be observed:

Ordered Steps (Numbered List): If the steps must be followed in a specific sequence, use a numbered list to present the order clearly. This ensures that readers understand the progression and importance of each step.

Example:

1. Clone repo.
2. cd into the cloned directory.
3. npm install

Unordered Steps (Bulleted List): If the order of the steps is not crucial to the outcome, use bulleted points. This provides flexibility for readers to approach the tasks as they prefer.

Example:

- Choose a color.
- Select a size.
- Identify a preferred style.

Adhering to these guidelines will ensure readers' clarity and ease of understanding, allowing them to follow instructions effectively, whether in a precise sequence or with more flexible options.

</details>

<details>

<summary>Capitalization</summary>

Key Point: Use standard American capitalization. Use sentence case for headings.

Follow the standard capitalization rules for American English. Additionally, use the following style standards consistently throughout the Hedera developer documentation:

Follow the official capitalization of Hedera products, services, or terms defined by open-source communities, e.g., Hedera Consensus Service, Hedera Improvement Proposal, and Secure Hashing Algorithm.

Capitalize each instance of network names mainnet, testnet, and mirrornet only when preceded by Hedera, e.g., Hedera Mainnet, Hedera Testnet, and Hedera Mirrrornet.

Do not use all-uppercase or camel case except in the following contexts: in official names, abbreviations, or variable names in a code block, e.g., HBAR, HIPs, or SHA384.

You should revise any sentence starting with lowercase word stylization to avoid creating a sentence with a lowercase word.

</details>

<details>

<summary>Oxford comma</summary>

The Oxford comma is the comma used immediately before the coordinating conjunction ("and" or "or") in a list of three or more items. In our written content, the use of the Oxford comma is required to maintain clarity and prevent ambiguity.

Example: "The team consists of product managers, developers, designers, and writers."

By consistently applying the Oxford comma, we ensure that the meaning of lists is clear, especially when individual items contain commas themselves. This standard reflects our dedication to ensuring clear and accurate communication in all of our documentation.

</details>

<details>

<summary>Abbreviations</summary>

Key Point: Use standard American and industry-standard abbreviations, e.g., NFT for non-fungible tokens. Avoid internet slang.

Abbreviations include acronyms, initialisms, shortened words, and contractions. In most contexts, the technical distinction between acronyms and initialisms isn't relevant; it's OK to use the phrase acronym to refer to both.

An acronym is formed from the first letters of words in a phrase/name but pronounced as if it were a word itself:

WAGMI for We're All Gonna Make It

DAO for Decentralized Autonomous Organization

An initialism is from the first letters of words in a phrase, but each letter is individually pronounced:

KYC for Know Your Customer

IPFS for InterPlanetary File System

A shortened word is just part of a word or phrase, sometimes with a period in the end:

Dr. for doctor

etc. for et cetera

Note: Some abbreviations can be acronyms or initialisms, depending on the speaker's preference—examples include FAQ and SQL. In some cases, the pronunciation determines whether to use a or an.

Long and short versions of a word [](#long-and-short-versions)

The short versions of the words are not abbreviations; if you use them, you don't need to put a period after them—for example:

application and app

synchronize and sync

If you're unsure whether a word is an abbreviation or a shortened version of a word, look in this list of resources. If that doesn't settle the issue, use the speaking test: if you speak the short version as a word (This is a demo version of the product), you can usually treat it as a word and not an abbreviation.

Don't create abbreviations [](#creating-abbreviations)

Use recognizable and industry-standard acronyms and initialisms. Abbreviations are intended to save the writer and the reader time. If the reader has to think twice about an abbreviation, it can slow down their reading comprehension.

Make abbreviations plural [](#making-abbreviations-plural)

Treat acronyms, initialisms, and other abbreviations as regular words when making them plural—for example, APIs, SDKs, and IDEs.

When to spell out a term [](#spelling-out)

In general, when an abbreviation is likely to be unfamiliar to the audience, spell out the first mention of the term and immediately follow with the abbreviation in parentheses, for example:

Miner Extracted Value (MEV)

elliptic-curve cryptography (ECC)

For all subsequent mentions of the term, use the abbreviation by itself. If the first mention of a term occurs in a heading or title, you can use the abbreviation and then spell out the abbreviation in the first paragraph that follows the heading or section title.

In some cases, spelling out an acronym doesn't help the reader understand the term. For example, writing out a portable document format doesn't help the reader understand what a PDF document is.

Note: The following acronyms rarely need to be spelled out: API, SDK, HTML, REST, URL, USB, and file formats such as PDF or XML.

</details>

glossary.md:

description: Hedera & Web3 Glossary - Comprehensive Guide for Developers

Glossary

This glossary intends to provide a reference for Hedera and general web3 key terms. The purpose is to assist developers, particularly those new to the field or non-specialists, in understanding essential definitions related to various aspects of this technology. It covers basic to complex concepts and essential development tools and is an accessible resource for developers.


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

\#

51% and 1/3 Attacks

A 51% attack, also known as a majority or double-spend attack, is a recognized threat in distributed networks like blockchains. This occurs when a miner or group gains control of more than half the network's mining power, thereby gaining the ability to disrupt transactions. While this is a widely acknowledged issue in the blockchain community, all DLTs are susceptible to 1/3 attacks when an attacker can establish a firewall.


A 1/3 attack is a less discussed threat in distributed networks such as Hedera, which occurs when over 1/3 of the network's nodes are compromised by bad actors using malicious code. This can potentially halt the acceptance of the correct block, making it impossible for others to identify the longest, correct chain and thereby preventing the processing and recording of valid transactions.

 For a more comprehensive breakdown of these attacks, check out Dr. Leemon Baird's talk at Harvard [here](#).

A

Account Alias

An account alias is a user-friendly identifier that can be classified as either a public key or an Ethereum Virtual Machine (EVM) address. It serves as a reference to the account object, in addition to its account number, and is assigned during the auto account creation process. The purpose of an account alias is to facilitate easier management and recall of accounts, particularly in distributed ledger technology and cryptocurrency contexts where addresses are often complex and difficult to remember. Instead of inputting a long string of characters, users can use simpler and more memorable aliases.

 For more information, refer to the Account Alias section on the Account Properties page.

Account ID

A unique identifier associated with an account on a distributed ledger. Each account on the ledger has a unique account ID, which is used to track and manage all transactions associated with that account.

Address

A unique identifier that represents a user or a destination on the network. It's similar to how an email address works: it's a location where cryptocurrency can be sent. In Hedera, an address could refer to an account ID associated with a specific user or smart contract.

Algorithm

An algorithm is a sequence of instructions that ensures a task or computation is completed. It can also predict results based on a specific input. Distributed ledger technology uses consensus algorithms to help reconcile a customer's bank account.

Anti-Money Laundering (AML)

A set of regulations and laws to help prevent crimes that produce monetary gain, such as tax evasion, fraud, selling stolen goods, drug and human trafficking, and corruption. Similar laws are being created for the blockchain industry to prevent money laundering activity.

Application Binary Interface (ABI)

The contract Application Binary Interface (ABI) is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction. Data is encoded according to its type, as described in this specification.

Asynchronous Byzantine Fault Tolerance (aBFT)

Asynchronous Byzantine fault tolerance (aBFT) is a property of Byzantine fault tolerant consensus algorithms, which allow for honest nodes of a network to guarantee to agree on the timing and order of a set of transactions fairly and securely. It's considered the highest degree of security in distributed systems.

In the context of Hedera, aBFT means that the network can reach consensus on the order and validity of transactions, even if some nodes aren't trustworthy or become compromised. This is achieved through the hashgraph consensus algorithm, which allows all nodes to agree on the order of transactions in a fair and secure way.

B

Balance Files

A snapshot of the state of all accounts within the Hedera Network at the time of their creation. These files, which are stored and made accessible by mirror nodes, include details such as account balances and account status information. They are created at regular intervals and can be used to independently verify the state of the network.

Bitcoin (BTC)

The first cryptocurrency based on a Proof of Work (PoW) blockchain. Bitcoin was created in 2009 by Satoshi Nakamoto – a pseudonym for an individual or group whose real identity is unknown – and the concept of cryptocurrency was outlined in a white paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System."

Blob

A large, unstructured data object. In Ethereum, blobs are introduced to enhance data availability for rollups (EIP-4844), but Hedera currently does not support blobs.

Block

A block is a batch of transactions that are linked together using cryptographic hashes. Each block contains a reference to its parent block, preserving the transaction history in a strictly ordered manner. Blocks are created approximately every 12 seconds to allow consensus on the network and contain a wide range of information.

In the context of Hedera, as per HIP-415, a "block" is a record file that contains all Record Stream Objects within a specific timeframe. Key properties of a block include the block number, block hash, and block timestamp. The concept of blocks is introduced to enhance interoperability with Ethereum Virtual Machine (EVM) based tools and platforms.

Blockchain

A type of distributed ledger technology (DLT) that maintains a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. The design of a blockchain is inherently resistant to data modification, making it secure and reliable for recording transactions across many computers. Please note that Hedera is a directed acyclic graph (DAG) and not a blockchain.

Block Hash

A 32-byte prefix of the running hash of the last Record Stream Object from the previous record file.

Block Number

A successive number assigned to each record file, incremented by one for each new file. For pre-existing networks, this value is bootstrapped through mirror nodes and subsequently maintained by service nodes.

Block Timestamp

The consensus timestamp of the first transaction in the Record file.

Bytecode

Bytecode is the information that Solidity code gets "translated" into. It contains instructions to the computer in binary. Bytecode is generally compact numeric codes, constants, and other pieces of information.

Compiled Bytecode: This is the result of compiling your high-level code (like Solidity for Ethereum smart contracts). This compiled bytecode is what the Ethereum Virtual Machine (EVM) can understand and execute. It includes everything your contract needs to run, including constructor logic, which is the code that runs when you deploy your contract.

Deployed Bytecode: After you deploy your contract to the blockchain, the deployed bytecode is what resides on the blockchain. It's a version of the compiled bytecode but without the constructor logic. The deployed bytecode represents the final, immutable code of your contract as it lives on the blockchain.

C

Call Trace

Contract call trace information captures the input, output, and gas details of all the nested smart contracts functions executed in a transaction. On Ethereum, these are occasionally called inner transactions but they simply capture snapshots of the message frame consideration the EVM encounters when processing a smart contract execution at each depth for all involved functions.

Central Bank Digital Currency (CBDC)

A digital currency that's issued by a central, nation-state-backed bank. The US Federal Reserve, the European Central Bank, and the Bank of England are examples of central banks. A nation's CBDC should be considered the digital equivalent of the nation's fiat currency.

Centralized Exchange (CEX)

A marketplace for buying, selling, and trading cryptocurrencies. A CEX is owned and operated by a centralized authority that maintains control over accounts and transactions. Some examples of a CEX are Coinbase and Crypto.com.

Chain

Another term for a blockchain.

Chain ID

A unique identifier for a specific chain within a network. In networks that support multiple chains or side chains, the chain ID helps distinguish between different chains.

Client

In the context of the Hedera SDK, a client is an object that allows you to interact with the Hedera Network by submitting transactions and queries. It is used to set up the operator account, configure the network, and set default transaction fees and query payments. Here is how to build a Hedera Client.

Cold Wallet

Sometimes referred to as a hardware wallet, and is a physical cryptocurrency device not connected to the internet. A cold wallet is considered to be more secure than a hot wallet.

Consensus

In the context of distributed ledger technologies (like blockchain and Hedera), consensus refers to the agreement of all nodes in the network on the validity and order of transactions. Hedera uses an algorithm called the Hashgraph Consensus Algorithm. Other consensus mechanisms include Proof of Work and Proof of Stake.

Contract Account

Contract accounts are accounts that are controlled by the code of a smart contract. They don't have a private key; instead, their behavior is determined by the smart contract's code. They can hold Ether and send transactions, but only in response to a transaction they received (i.e., as part of the execution of a smart contract's code).

Cryptocurrency

A type of digital or virtual currency used for payment transactions. Unlike fiat currencies, such as the U.S. Dollar, it is not issued or controlled by governments or financial institutions. Instead, it is monitored on a peer-to-peer network, such as a distributed ledger.

Crypto Faucet

A platform, application, or website that rewards users in cryptocurrency when they complete certain tasks. Rewards tend to be in tiny increments of cryptocurrency. Rewardable tasks can include browser mining, playing games, watching videos, completing surveys, referring new users, etc.

Cryptography/Cryptographic

Cryptography is the art of disguising data, so that only the intended recipient can read it. Encryption and decryption are the main components of cryptography. In blockchain technology, cryptography offers security by ensuring transactions between nodes are encrypted.

The use of cryptography in conjunction with distributed networks allows for these networks to be both public and secure. Each DLT address generated for a user is paired with a private key, which allows the user to send and receive

transactions with that address.

Crypto Wallet

A form of digital wallet designed for web3. Crypto wallets help you manage permissions with whom you share data, store, and send cryptocurrency, NFTs, and more. Your crypto wallet contains a private key that identifies and assesses the assets that are yours. In this way, you can think of a wallet like MetaMask or HashPack as a digital identity management system.

Examples: non-custodial wallet, hardware wallet, custodial wallet

Custodial Wallet

A wallet used to digitally store fiat and cryptocurrencies. A trusted third party (custodian) is empowered with their own private key to manage the wallet for the owner of the wallet holder. For example, the custodian can send and receive payments at the owner's request.

Custody

Custody refers to the secure storage of digital assets by a specialized provider on behalf of clients. It involves safeguarding private keys and providing insurance, audits, and reporting to prove holdings.

D

Decentralization

A fundamental concept in distributed ledger technology that refers to the distribution of power and decision-making across a network or system rather than being controlled by a single entity or authority. Decentralization is a key feature that allows for trustless and transparent transactions without intermediaries.

Hedera, for example, is governed by a decentralized council of diverse organizations. A distributed network of nodes processes transactions on the Hedera Network, and the source code for the Hedera protocol is open review.

Decentralized Application (DApp)

Decentralized applications that run on peer-to-peer distributed ledger technology networks rather than a centralized server. They are sometimes referred to as web3 applications.

Decentralized Autonomous Organization (DAO)

A member-owned group or organization that operates without centralized leadership using distributed ledger technology. Instead of a centralized authority, a DAO is owned and governed by community members (token holders). A DAO's financial transactions and rules are encoded in smart contracts and recorded on a distributed ledger. The token-holders vote on any changes to the

rules or organizational structures, and voting power is typically distributed across users based on the number of tokens they hold. All DAO activity is transparent and fully public.

Examples: HSuite, Developer DAO

Decentralized Exchange (DEX)

A marketplace for users to buy, sell, and trade cryptocurrencies without a centralized intermediary to provide liquidity and verify transactions. Instead, other users provide liquidity, and transactions are verified through the distributed ledger.

Examples: SaucerSwap, HeliSwap, Pangolin

Decentralized Finance (DeFi)

A peer-to-peer financial technology built on distributed ledgers without intermediaries. DeFi services cover lending, borrowing, trading, derivatives, insurance, and prediction markets. Services are decentralized and not controlled by a single authority, such as those from traditional banking systems.

Examples: Stader Labs, hashport

Decentralized Identifiers (DIDs)

W3C Decentralized Identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID. DIDs are URIs that associate a DID subject with a DID document allowing trustable interactions associated with that subject.

Directed Acyclic Graph (DAG)

A structure used for data organization and representation of associations using circles and lines. DAGs are good for mapping an efficient process and visualizing relationship flows, such as family trees, and are viable replacements for blockchains due to their speed and data-storage capabilities. Hedera is a DAG and not a blockchain.

Distributed Denial-of-Service (DDoS)

A Distributed Denial-of-Service (DDoS) attack is a malicious attempt to disrupt the normal operations of a network, service, or website by overwhelming it with a flood of internet traffic from multiple sources.

In the context of Hedera, a DDoS attack would involve an attempt to overwhelm the network with a flood of transactions or requests, aiming to disrupt its normal operation. However, due to Hedera's unique consensus mechanism and its use of asynchronous Byzantine Fault Tolerance (aBFT), it is designed to be resilient to such attacks.

Distributed Ledger

A distributed ledger is a database shared by multiple participants in which each participant maintains and updates a synchronized copy of the data. Distributed ledgers allow members to securely verify, execute, and record their own transactions without relying on an intermediary, such as a bank, broker, or auditor. A distributed ledger can also be called a DLT, the acronym for Distributed Ledger Technology.

Distributed Ledger Technology (DLT)

A technology that allows the existence of distributed ledgers, such as blockchains and DAGs, that use distributed ledgers stored on separate, connected devices in a network to ensure data accuracy and security. Unlike traditional databases, distributed ledgers have no central data store or administration functionality.

Double Spend Problem

Spending the same coins more than once, especially at the same time. This is one of the primary "attacks" distributed ledger networks have been designed to resist.

E

Elliptic Curve Digital Signature Algorithm (ECDSA)

A cryptographic algorithm for digital signatures is a crucial component in distributed ledger technology. It is a variant of the Digital Signature Algorithm (DSA) that operates on elliptic curve cryptography. Elliptic curve cryptography, and thus ECDSA, offers a high level of security with relatively small keys, which makes it more efficient than some other cryptographic systems. As a result, ECDSA is widely used in many systems that require secure digital signatures, including Bitcoin, Ethereum, and other distributed ledger networks.

ECDSA secp256k1

A cryptographic algorithm used for data integrity and widely used in cryptocurrencies. In this term, "secp256k1" refers to the specific parameters of the elliptic curve used, making it efficient for computations. Hedera supports this signature scheme for generating cryptographic keys and signing transactions.

Ed25519

In the context of Hedera, Ed25519 is one of the signature schemes supported for the creation and verification of digital signatures. It's often used for signing transactions, as it provides a strong level of security while still being efficient to compute.

EIP (Ethereum Improvement Proposal)

A standard for proposing changes and new features to the Ethereum protocol. Some EIPs are relevant for Hedera's EVM compatibility and can be referenced

here.

ERC-20

A technical standard for fungible tokens created using the Ethereum blockchain. A fungible token is interchangeable with another token—where the well-known non-fungible tokens (NFTs) are not interchangeable. ERC-20 allows developers to create smart-contract-enabled tokens that can be used with other products and services. These tokens represent an asset, right, ownership, access, cryptocurrency, or anything else that is not unique in and of itself but can be transferred. Check out the details of the ERC-20 standards [here](#).

Hedera supports the creation and management of custom tokens through the Hedera Token Service (HTS).

ERC-721

A type of token standard – a template or format that other developers agree to follow. Following the same standards makes writing code easier, more predictable, and reusable. Each ERC-721 token is unique and not interchangeable with any other token - hence the term non-fungible. This unique characteristic allows ERC-721 tokens to represent ownership of unique items like a particular piece of real estate or a specific piece of art. Check out the details of the ERC-721 standard [here](#).

Hedera supports the creation and management of both fungible and non-fungible tokens using the ERC-721 token standard.

Ether (ETH)

The native cryptocurrency for Ethereum.

Ethereum

An open-source, decentralized blockchain platform with smart contract functionality. Ether (ETH) is the native cryptocurrency of Ethereum. Developers can use Ethereum or an Ethereum Virtual Machine (EVM) to create and run decentralized applications containing smart contract functionality, as well as issue new crypto assets, known as tokens.

Ethereum Virtual Machine (EVM)

The runtime environment in Ethereum, where smart contracts are executed. It is a Turing-complete virtual machine that executes scripts used to implement certain operations on the Ethereum blockchain. The EVM is contained within the client software needed to run a node on Ethereum, and it manages the state of the blockchain and enables smart contract functionality.

Event Files

Event files record the history of transactions that have occurred on the Hedera Network. They contain the details of all the transactions and the order in which

they were processed. The files are created on each node in the network and are then shared with mirror nodes, which are nodes that maintain a copy of the network's history but do not participate in consensus.

Externally Owned Account (EOA)

An account controlled by private keys that can send transactions to other accounts or deploy smart contracts onto the network. An EOA is created by generating a public-private key pair. Transactions sent from an EOA are initiated by an external actor who holds the private key. Some popular web3 wallets (e.g., MetaMask, HashPack) are EOAs.

F

Fair Order

The principle that all transactions on a distributed ledger should be treated equally, without preferential treatment and describes the consistency of records for the transaction queue of a ledger. Transactions in fair order are recorded on a ledger consistent with the order in which they are transmitted.

Fallback Fee

A "fallback fee" in the context of Hedera and smart contracts is a type of custom fee that is charged to the recipient of a token transfer, rather than the sender. This can lead to vulnerabilities in smart contracts if they unknowingly accept a token with a high fallback fee, resulting in a loss of funds. Discussions on mitigation strategies, such as sender-pays models or limits on custom fees, are ongoing. Here's a comprehensive blog post covering fallback fees.

Fee Collector Account

A fee collector account is an account designated to receive transaction fees and custom token fees on the Hedera network. The account collects the fees users pay for executing transactions and queries on the network.

Fiat Currency

A term widely used across the financial industry (and even beyond) to refer to government-backed national currencies such as the U.S. dollar or the British pound. Even so, the concept of fiat currency is deeply intertwined with multiple aspects of distributed ledger technology and cryptocurrency usage. One of those aspects concerns how individuals and organizations onboard themselves into the worlds of DLTs and cryptocurrency.

Finality

The assurance or guarantee that completed transactions or blocks can't be reversed, revoked, canceled, or changed in any way. The latency level of a ledger will ultimately affect the chain's finality rate.

Fixed Fee

A fixed fee refers to a predetermined amount of tokens or HBAR that is transferred to a specified fee collection account each time a token transfer occurs. Fixed fees are one of the types of custom fees you can define when creating a token on Hedera.

Fork

A fork is an event in which a blockchain splits into two separate chains. A fork occurs when software updates to its functionality are introduced, but not all participants (miners, developers) agree on them.

Fractional Fee

A fractional fee is a type of custom fee that can be set when creating a token on the Hedera network. Fractional fees are calculated as a fraction of the total value of the tokens that are being transferred in a transaction.

Fungible Token

A token or digital asset that is equal in value and is mutually interchangeable with assets of the same type. For example, a Bitcoin is always valued the same as another Bitcoin.

G

Gas Fee

The cost of performing a transaction or smart contract operation.

Gas Limit

The maximum amount of gas a sender is willing to spend in a transaction.

Genesis Block

The first block of data that is processed and validated to form a new blockchain, often referred to as block 0 or block 1.

Gossip about Gossip

The history of how events are related to each other through their parent hashes in a hashgraph. It expresses itself as a directed acyclic graph, a graph of hashes, or a hashgraph. The hashgraph records the history of how members communicated and grow directionally over time as more gossip syncs take place and events are created. All members keep a local copy of the hashgraph, which updates as members sync with one another.

Gossip Protocol

A protocol for communication between network nodes in a distributed, peer-to-peer network and sometimes used in distributed systems such as ledgers, databases, and file systems because they are secure, reliable, fault-tolerant, and efficient & decentralized.

Gossip protocol is based on the construct of "gossiping", or spreading information from one node to another. Every few seconds, each node sends a message to a random node in the network about itself and other nodes, then that message moves from node to node until its communication reaches across the entire network.

Several DLTs, such as blockchains, utilize the Gossip protocol. It is used in Hedera (along with Gossip about Gossip) for nodes to distribute transaction-related data. The Bitcoin network uses the Gossip protocol to broadcast block-mining solutions. It is also used in the Hyperledger Fabric network and Ethereum, among others.

Governance

The system of proposing, managing, and implementing changes to distributed ledgers, including management of transaction fee allocation, user interface changes, developer recruitment, licensing, product roadmaps, development fund distribution, and other policies.

<details>

<summary>There are two types of DLT governance: on-chain and off-chain</summary>

On-chain governance: stakeholders can vote on changes via rules encoded into the blockchain protocol. Stakeholders typically hold governance tokens, including miners, developers, and investors. Developers propose changes through code updates, and each node votes to accept or reject the proposed changes. Voting mechanisms for on-chain governance vary but likely include smart contract functionality. Decentralized Autonomous Organizations (DAOs) utilize on-chain governance to run operations without a central authority.

Off-chain governance: changes are made in online forums, social media, conferences, email groups, and other public spaces. Off-chain governance is successful when all stakeholders agree and make all updates and implementations in unison. If consensus cannot be reached, the network can split, or fork, into two chains running different versions of the software, and the chain with the most transactional hashing power is considered to be the successor to the original chain.

</details>

Governance Token

A governance token is a utility token that can help democratize decision-making related to managing a public ledger, decentralized apps, and other decentralized protocols. One governance token represents one vote in any decision-making process where the outcome is determined by the option that registers the most votes.

Hardhat

A development environment for Ethereum software. It consists of different components for editing, compiling, debugging, and deploying smart contracts and dApps, all working together to create a complete development environment. Learn more about it [here](#).

Hash

A mathematical function in which an input of a string of information (numbers, letters, media files) is output into a fixed size. Hash functions are used for data integrity, generating unique identifiers, digital signature creation, and consensus mechanisms in DLTs.

Hashgraph

A Hashgraph, in the context of Hedera, is a way to chart events created by members in order to achieve consensus order and consensus timestamps, where an event is a circle: a container for a blockchain transaction, and members are full nodes.

Hashrate

A metric of cryptocurrency computing power determined by measuring how many hashes are generated by all the miners who are simultaneously trying to solve the current block or any given block on a mined distributed ledger such as Bitcoin.

HashScan

HashScan is a ledger explorer and analytics platform for the Hedera network. It allows users to easily search for transactions, as well as provide information about each transaction.

HBAR

The native cryptocurrency of the Hedera Network.

Hedera Consensus Service (HCS)

This service enables developers to create a verifiable timestamp and order of events for any application. In other words, it allows any application to send messages to the Hedera Network and receive consensus timestamps and fair order. This can be useful for various applications, including supply chain tracking, fair order for marketplaces, and distributed systems coordination.

Hedera File Service

The Hedera File Service provides a decentralized file storage platform that allows developers to securely store and access files on a distributed network of computers using hash as a file identifier.

Hedera Improvement Proposal (HIP)

A Hedera Improvement Proposal (HIP) is a proposal that can range from core protocol changes to the applications, frameworks, and protocols built on the Hedera public network and used by the community. HIPs are reviewed and evaluated by the Hedera Council, core developers, and editors.

Hedera Smart Contract Service (HSCS)

The Hedera Smart Contract Service (HSCS) is a service provided by the Hedera network that integrates the features of Hedera's native entity functionality with a highly optimized and performant Ethereum Virtual Machine (EVM). It allows developers to deploy and interact with smart contracts on the Hedera network, offering high throughput, fast finality, predictable and affordable fees, and fair transaction ordering.

Hedera Token Service (HTS)

This service provides the ability to issue and manage tokens on the Hedera Network. With the Hedera Token Service, users can define, mint, burn, and configure tokens without deploying a smart contract. It supports both fungible tokens (like ERC-20 tokens) and non-fungible tokens (like ERC-721 tokens). The HTS is designed to be fast, secure, and efficient, with low fees and finality of transactions.

Hot Wallet

A cryptocurrency wallet that is always connected to the internet and can be either a mobile application, web application, or browser extension. As they are always connected to the Internet, hot wallets are not as secure as cold wallets, which are only connected to the Internet during transactions.

Hyperledger Besu EVM

An open-source Ethereum client developed under the Hyperledger project and a virtual machine that replaced the EthereumJ virtual machine in the Hedera Services release 0.19 as a result of HIP-26. This migration enables Hedera to maintain parity with Ethereum Mainnet evolutions, such as the EVM container formats, new opcodes, and precompiled contracts. The Besu integration is configured to use the "London" hard fork of Ethereum Mainnet.

I

Immutability

The property of a distributed ledger technology, such as blockchain that prevents data from being altered or tampered with. Once cryptographically

secured and stored, the ledger record cannot be altered. Because each hash on each block in a chain is unique, data cannot be tampered with after it is logged. This creates an ultra-high-integrity record of information about the whole blockchain.

Initial Coin Offering (ICO)

Initial Coin Offering, or ICO (also stands for Initial Cryptocurrency Offering), is a funding opportunity for a new cryptocurrency coin, application, or service. Unlike IEOs and IDOs, tokens are purchased from a project website, not an online exchange platform.

Interoperable

Interoperability on distributed ledgers means the ability to connect and communicate (exchange & comprehend data) with other distributed ledger networks, creating a pathway to the easy exchange of assets. (In a broader sense, interoperability refers to the power of communication between systems.) Interoperability protocols allow a distributed ledger to read and write messages to other networks, easily exchanging data.

InterPlanetary File System (IPFS)

InterPlanetary File System (IPFS) is a hypermedia protocol and peer-to-peer network for storing and sharing data files. Unlike HTTP, an IPFS network does not require a host server. Instead, IPFS is decentralized and stores the information on hundreds of thousands of nodes (servers) worldwide. IPFS was created by Juan Benet of Protocol Labs and launched in February 2015.

ISO 20022

ISO 20022 is a global standards scheme for financial industry messaging which includes methodology, process, and repository standards. It covers financial information transferred between financial institutions that includes payment transactions, securities trading and settlement information, credit and debit card transactions, and other financial information.

Currently, seven DLTs adhere to ISO 20022 standards: Algorand (ALGO), Hedera (HBAR), IOTA (MIOTA), Quant (QNT), Ripple (XRP), Stellar (XLM), XDC Network (XDC).

J

JSON-RPC Relay

An open-source project implementing the Ethereum JSON-RPC standard. The Hedera JSON-RPC relay allows developers to interact with Hedera nodes using familiar Ethereum tools. This allows Ethereum developers to deploy, query, and execute contracts on the Hedera Network as they would on Ethereum.

K

Know Your Customer (KYC)

A standard for the investment industry in which an investor's identity can be verified and evaluated for business relationships. Players in the blockchain industry must heed to KYC standards because cryptocurrency exchanges require them for Anti-money laundering (AML) compliance.

L

Liquidity Pool

A collection of shared cryptocurrency coins/tokens locked under a smart contract, ensuring liquidity that allows participants to trade easily. They are used for trading in decentralized exchanges (DEX), and for providing liquidity to decentralized finance (DeFi) protocols, which makes them an important part of yield farming.

 M

Mainnet

Mainnet (main network) is a distributed ledger network that is developed, tested, and fully deployed for public use. In contrast, a testnet (test network) is a distributed ledger network used by developers to test smart contracts and dApps before they are deployed to the live mainnet.

Mempool

An Unordered Transaction Pool or Memory Pool, also referred to as mempool (a combination of the words memory and pool), is a list of cryptocurrency transactions that have not yet been processed by a node. Once a transaction is added to a block, it disappears from the mempool. The larger the mempool size, the more congestion in network traffic, and the longer the confirmation time, which results in higher transaction fees. Hedera uses aBFT consensus (aBFT) algorithms, and the most important aspect of aBFT networks is that there is no memory pool of transactions. Learn more about it [here](#).

Merkle Root

Merkle root is a mathematical method of confirming Merkle tree hashes. It is the root hash of all other hashes in the Merkle tree.

Merkle Tree

A Merkle tree, a hash tree, is a data structure used in distributed ledger technology for efficient data verification and transfer on peer-to-peer networks. It consists of leaf nodes (hashes of crypto transactions in a block), non-leaf nodes (hashes of leaf nodes), and the Merkle root (the final hash). Merkle Trees enhance data integrity and enable Simplified Payment Verification (SPV), allowing wallets to verify transactions without needing the entire distributed ledger network.

MetaMask

A non-custodial wallet used to interact with Ethereum and other EVM-compatible networks. It allows users to access their wallets through a browser extension or mobile app, which can then be used to interact with decentralized applications. Learn more about it [here](#).

Mint

The creation of a token on a distributed network. Most often used in the context of non-fungible tokens (NFTs).

Mirror Nodes

A mirror node is used to store and cost-effectively query historical data from the public ledger while minimizing the use of Hedera Network resources. Mirror nodes support the Hedera Network services currently available and can be used to retrieve transactions and records, event files, and balance files.

Multisig

The requirement for a transaction to have two or more signatures before it can be executed, often used in DAOs. Multisig provides more security than single-signature transactions.

N

Native Cryptocurrency

Native cryptocurrency is the digital currency inherent to a DLT such as a blockchain. For some examples, Bitcoin (BTC) is the native cryptocurrency for the Bitcoin blockchain, or Hedera (HBAR) for Hedera.

Network Explorer

A network explorer, sometimes called a blockchain or ledger explorer, is an online tool or application that allows users to browse and search the blocks, transactions, addresses, and other data on a distributed network.

Node

In web3, a node is any participating computer in a peer-to-peer network that is propagating the verification of transactions, creating blocks, and/or maintaining the distributed network.

Non-Custodial Wallet

A non-custodial wallet is a decentralized wallet where the user has complete control over their private keys, allowing them to store and manage their digital assets. Ledger, Exodus, HashPack, and MetaMask are examples of popular non-

custodial wallets.

Non-Fungible Token (NFT)

A unique digital asset with ownership rights that are stored on a distributed network. An NFT can be a one-of-a-kind image, video, composed music, game asset, medical record, event ticket, domain, or other creative media that is tokenized, therefore, can be bought, sold, or traded on a distributed ledger using various cryptocurrencies.

Nonce

A cryptographic nonce is an arbitrary, single-use, whole, binary number used for communication in security functions. The acronym "nonce" is an abbreviation of "number only used once." A basic nonce in Bitcoin is 32-bit (4-byte). A strong nonce has at least 128 bits of entropy. Solving the hash on a block is how a miner finds the nonce.

Uses for a nonce: authentication, initialization vectors, hashing, indexing, smart contracts, and block validation.

Within a distributed ledger, a nonce is proof of work for an encrypted (or hashed) block. Transaction verification and other data contained within that block can then be verified via the added nonce.

0

Off-Chain

Off-chain refers to transactions that occur off of the distributed ledger. These transactions can be peer-to-peer or through a third-party intermediary, and they are not subject to the DLT's protocols. As a result, they can be faster and cheaper than on-chain transactions but might not offer the same level of security or decentralization. Off-chain transactions can also be private, as they are not necessarily visible to all nodes in the network.

Off-Chain Storage

Off-chain storage is a provided service for storing data that cannot be stored on a ledger and is useful for off-chain data, such as real-world data, data that is required to be changed or deleted, and data that is too large to store on-chain efficiently.

On-Chain

On-chain refers to transactions that occur on the distributed ledger, and as a result, they are fully subject to the DLT's protocols. This includes the consensus protocol, and therefore these transactions are public, immutable, and auditable. The ledger secures on-chain transactions and are visible to all nodes in the network.

Opcodes

Short for "operation code," an opcode is a single instruction that the Ethereum Virtual Machine (EVM) can execute. Each opcode performs a specific operation, such as arithmetic calculations, data storage, or control flow management. Opcodes are the low-level building blocks of EVM smart contracts written in Solidity or other EVM-compatible languages.

Oracles

In distributed ledger technology, oracles are third-party services that connect distributed ledger (on-chain) data to data coming from external (off-chain) systems. With this function, oracle technologies can provide the off-chain data needed to meet the conditions of a smart contract.

Ordinals

Bitcoin Ordinals is a protocol that allows individual satoshis (SATS) in a Bitcoin blockchain to be assigned a unique identifier and transacted with extra data attached. This protocol became the foundation for a unique collaborative venture between Hgraph, a developer tooling and Web3 consulting entity, and Turtle Moon, a pioneer in NFT services. Together, they orchestrated the first token-gated vote on Bitcoin using the H4NGRY's Ordinals NFT collection titled "Kid Pepes." Every vote cast during this initiative was instantaneously recorded on Hedera, serving as the trust layer of the voting system, with the final vote outcomes being inscribed back onto Bitcoin.

The integration showcases a groundbreaking application of distributed ledger technology by initiating the voting process on Bitcoin, where ballot details including start and end times, a unique identifier, and a Hedera Consensus Service topic ID are inscribed. This process transitioned to Hedera for recording votes, offering a transparent, tamper-proof voting system. The final ballot results, which are recorded back on Bitcoin, can be validated against the original data points, embodying a robust, transparent, and innovative use of blockchain technology for democratic processes. Read more about this integration [here](#).

P

Peer-to-Peer (P2P)

A decentralized network interaction model where individual nodes ("peers") connect directly with each other instead of through centralized servers or authorities.

In a peer-to-peer network, each node can act both as a client and as a server. This contrasts with the traditional client-server model, where client nodes request resources or services, and server nodes fulfill those requests.

Permissioned

Permissioned DLTs are private distributed ledgers that require user access from an entity (or entities) that control the network.

Permissionless

Permissionless means a system or property accessible by anyone without permission from a central authority. Permissionless ledgers are public distributed ledgers, including public blockchains, that allow anyone to join at any time, to read, write, or participate in the ledger network. There are no entities that regulate or control the network or its users.

PostgreSQL

A free and open-source relational database management system emphasizing extensibility and SQL compliance. Mirror nodes use a PostgreSQL database to store the transaction and event data organized in a structure that mirrors the Hedera Network.

Precompile

In Ethereum and other smart contract platforms, precompile (or precompiled contract) is a special kind of smart contract with a fixed address that is implemented in the blockchain client itself rather than in the Ethereum Virtual Machine (EVM).

The main advantage of precompiles is that they can be executed more efficiently than regular smart contracts. This is because they are written in low-level code and don't need to be interpreted by the EVM. This makes them useful for computationally intensive operations, such as cryptographic functions or mathematical operations.

For example, Ethereum includes precompiled contracts for operations like elliptic curve multiplication and pairing, as well as for hash functions like SHA-256.

Private Key

A private key is an alphanumeric string of data that corresponds to a single specific account in a wallet. Private keys can be thought of as a password that grants access to, and control over, that specific crypto account. Never reveal your private key to anyone, as whoever controls the private key controls the account. If you lose your private key, you lose access to that account.

Proof-of-Reserves (PoR)

A Proof of Reserves (PoR) is an independent audit conducted by a third party that seeks to ensure that a custodian holds the assets it claims to on behalf of its clients.

Proof-of-Stake (PoS)

A consensus mechanism in which an individual runs distributed ledger software (a "consensus node") responsible for validating transactions, blocks, and the state of the network.

The individual must first "stake" an amount of cryptocurrency, such as ether, in a smart contract; this allows them to participate in consensus-building. If the individual's consensus node functions within specifications, they are rewarded,

usually with cryptocurrency. If, on the other hand, the node functions poorly or maliciously, the staked tokens can be "slashed" or taken away.

PoS requires a negligible amount of computing power compared to Proof of Work consensus.

Proof-of-Work (PoW)

Proof of Work (PoW) is a consensus mechanism for confirming transactions and adding new blocks to blockchains.

Here's how it works: To create a new block, a cryptocurrency miner competes with other miners to solve a cryptographic puzzle. This is done by generating a cryptographic hash that needs to match the target hash for the current block. The first miner that correctly generates the hash can add the block and once the hash solution is verified, receives block rewards in the form of cryptocurrency.

When the puzzle gets solved and verified, PoW consensus is established, as the solution itself "proves" that "work" was done to solve it.

Protocol

In the blockchain industry, the word "protocol" refers to any distributed ledger-based service, including the blockchains themselves and any services or applications that run on them, that can programmatically receive and respond to specially formatted requests.

For example, if a public ledger can programmatically (via software as opposed to a user interface for humans) receive a request (and respond to it accordingly) to transfer cryptocurrency from one ledger account to another (which most chains can do), then it's a protocol.

Proto-Danksharding

A simplified version of dank sharding aimed at improving data availability without the full complexity of sharding.

Public Key

Cryptography and the modern software tools built with it often center around a key pair: public and private. You can derive a public key from a private key, but you cannot derive a private key from a public key.

In messaging, the public key is obtained and used by anyone to encrypt messages before they are sent to a known recipient with a matching private key for decryption.

By pairing a public key with a private key, transactions can be sent in public with no fear of someone "hacking" or altering them: the public key has encrypted the transaction into a format unreadable by anyone except the intended party and the intended party only.

Q

Queries

Queries are requests processed only by the single node to which they are sent. Clients send queries to retrieve some aspect of the current consensus state, like the balance of an account. Certain queries are free, but generally, queries are subject to fees.

Quorum

The minimum number of members required to vote on a proposal before it can be accepted, ensuring decisions are made with sufficient participation.

R

Record File

A record file is a file that contains the details of a transaction that occurred on the Hedera Network. It provides greater detail about the transaction than a receipt, such as a consensus timestamp it received or the results of a smart contract function call. Hedera Mirror Nodes store the record file and can be accessed through the mirror node REST API.

Remix IDE

The Remix IDE is a user-friendly platform that allows you to easily write and compile your smart contracts and perform other tasks such as debugging and testing. Learn more about it [here](#).

REST API

REST APIs are used for interactions with services provided by Hedera, such as account balance checks, transaction submissions, and other actions related to the consensus service, token service, and file service.

Hedera's REST APIs allow developers to easily interact with the Hedera Network without needing to directly connect to the network's nodes. Instead, they can send HTTP requests to a server that acts as an intermediary between them and the Hedera Network, simplifying the process of building applications on top of Hedera.

Royalty Fee

A royalty fee refers to a specific type of custom fee that is applied to the transfer of non-fungible tokens (NFTs). A royalty fee is essentially a way to ensure that the original creator of an NFT continues to earn revenue from it even after the initial sale, as it allows them to earn a fraction of the sales every time the NFT is bought or sold in the future.

S

Security Model

A security model outlines network security measures designed to protect a network from threats and ongoing attacks. In distributed ledger technology, a security model includes consensus mechanisms, cryptographic techniques, permissions, and smart contract security.

For example, Hedera's security model relies on the unique Hashgraph consensus algorithm that ensures fairness, security, and speed. It utilizes digital signatures for transaction verification, and it operates a permissioned network with trusted entities as nodes. Hedera also provides services like the Hedera Token Service (HTS) and Hedera Smart Contract Service (HSCS), built with security considerations in mind.

Shard

A shard is a wedge of a blockchain network. Shards contain tokenized digital assets; a shard is defined by the contained asset and by the properties of the digital asset. Shards function as a distributed ledger: sharing data between shards is a cornerstone of their build.

Smart Contract

A smart contract is a program consisting of a set of logic (state variables, functions, event handlers, etc.) or rules that can be deployed, stored, and accessed on Hedera. The functions within a smart contract can update and manage the state of the contract and read data from the deployed contract. Smart contracts are secure, tamper-proof, and transparent, offering a new level of trust and efficiency.

SDK

A Software Development Kit (SDK), sometimes called a devkit, is a set of tools provided to software developers for building software applications. Tools can include Application Programming Interfaces (APIs), Integrated Development Environments (IDE), frameworks, code libraries, debuggers, code samples, test projects, and documentation.

Solidity

Solidity is a Turing-complete object-oriented programming language for developing smart contracts on the Ethereum Virtual Machine (EVM) and other compatible distributed networks. It supports various functions necessary for operating decentralized applications and systems, including facilitating transactions, voting, multi-signature wallets, creating dApps, and crowdfunding. To learn more about the Solidity programming language, check out the documentation maintained by the Solidity team [here](#).

Soul-Bound Token (SBT)

Soul-bound tokens, also known as non-transferrable NFTs (non-fungible tokens), are digital assets permanently tied to a specific individual (or crypto wallet) – meaning they cannot be transferred to others.

Examples of SBT uses include attendance verification for events, job achievement certification, digital identification storage, membership credentials, medical

records management, and reputation-based voting for DAO governance models.

Source Code

Source code, which software developers program, is data, metadata, and data schema that collectively form the basis of any given ledger's transactional content. Source code is used to automate the business processes and algorithms behind the operation of a given distributed network (e.g., Bitcoin, Ethereum, Hedera, etc.).

Sourcify

Sourcify is a decentralized Solidity source code and metadata verification tool and repository and acts as a base layer allowing other tools to build on top of it. Sourcify enables transparent and human-readable smart contract interactions through automated Solidity contract verification, contract metadata, and NatSpec comments. More info on [Sourcify.dev](https://sourcify.dev).

Stablecoin

A stablecoin is a specialized form of cryptocurrency engineered to maintain a stable value by pegging it to an external asset, such as a fiat currency like the US Dollar or a commodity like gold. By doing so, stablecoins seek to combine the programmability and ease of transfer inherent in cryptocurrencies with the price stability typically associated with traditional currencies. This low-volatility makes stablecoins particularly useful for transactions, cross-border payments, and as a stable asset within decentralized finance ecosystems. They essentially serve as a bridge between the conventional financial infrastructure and the emerging crypto-economic landscape, enabling fiat currencies to exist in a format that can be transferred with greater fluidity and efficiency across distributed networks.

Staking

The process of participating in a proof-of-stake system to validate transactions and earn rewards. When staked, coins are locked but can be unlocked for trading. Staking allows participants (stakeholders) to earn rewards on their holdings, typically in tokens or coins.

State Machine

A state machine represents the state of the Hedera Network, including the balance of every account, the contents of any files stored via the Hedera File Service, and the details of any tokens created via the Hedera Token Service. Transactions in Hedera cause the state to transition from one state to another, and the Hedera Network achieves consensus on the order of these transactions and their resulting state transitions.

State Proof

State proofs are a cryptographically secure, transferable, and storable mechanism for enabling trust and confidence in representations of the state of

Hedera. The fundamental value proposition of a distributed ledger is that participants need not have special trust in any single node maintaining the state. Hedera's state proofs ensure that clients querying the state can be given the necessary confidence that any data, even if returned by a single node, accurately represents the consensus state maintained by the full network.

System Smart Contracts

System smart contracts are smart contracts that are implemented and maintained on the Hedera network as part of the core codebase that other contracts can invoke. These contracts have a permanent contract address in the Hedera network, meaning they will always be available at the same contract address.

T

Testnet

In the context of Hedera, the Hedera Testnet is a network used by developers for testing their applications before deploying them on the Hedera mainnet. It allows for testing and debugging in a controlled environment, ensuring that any potential issues are resolved before the application is launched for real use.

Timestamp

A timestamp is a piece of recorded data that verifies the data existed at a specific date and time. In distributed ledger technology, a timestamp reveals when a block has been mined and validated. With the Bitcoin blockchain, timestamps help solve the double spending problem.

Token

A token generally refers to a type of cryptocurrency representing an asset or a specific use and resides on its own blockchain. Tokens can represent any fungible and tradable assets, from commodities to loyalty points to even other cryptocurrencies.

Tokenization

Tokenization is the process of underwriting the value of a token with a real-world tangible, such as gold, real estate, art, a stamp collection, etc.

Transaction

In distributed ledger technology, a transaction (tx or TX) refers to an exchange of cryptocurrency on any distributed ledger network. The use of the ledger system ensures a record of all transactions. The speed of completion of the exchange depends on several variables, such as which network is used, traffic at the time of the exchange, the size of the transaction fee, etc. An exchange can be completed in as little as a few seconds or minutes or several hours, depending on these variables.

Transaction Fee

A fee associated with a transaction that compensates the Hedera network for processing and maintaining the transaction in a consensus state.

Transactions Per Second (TPS)

The number of transactions per second (TPS) a distributed network can process.

Trustless

Trustless is a term associated with distributed ledger technology such as blockchain-based networks because participants don't need the support of a "trusted" central party, such as a bank, broker, lawyer, or other centralized authority, to facilitate the validation, execution, and journaling of transactions between multiple parties.

Turing Complete

Turing complete means a system or a language has the ability to simulate any computer algorithm, no matter how complex. It's like saying if you give this system enough time and memory, it can run any program a computer can execute, whether it's a simple calculator or a complex videogame. The importance of Turing completeness as it pertains to distributed ledger technology is that it allows for the programmatic development of smart contracts.

V

Validation

A distributed network maintains its integrity and security through validation. The nodes in the network process transactions before adding them to the ledger. This includes checking that the transaction is properly signed, that the sender's account has enough HBAR to cover the transaction and its fees, and that the transaction doesn't conflict with the current state of the ledger. By ensuring that all transactions adhere to the network's rules, validation helps prevent fraud, double-spending, and other forms of abuse.

Virtual Machine

A virtual machine (VM) is a software representation of a computer and VM technology related to distributed ledger technology as a platform for digital transformation, business innovation, and industry disruption. A VM can run software that's independent of the underlying machine (aka the host) that it runs on.

Virtual Merkle Tree

A virtual Merkle tree, as proposed in HIP-25, is a virtualized on-disk data structure designed to support the scaling of billions of entities per node on the Hedera Network. It aims to optimize memory usage by moving much of the system state out of RAM and onto disk, enhancing the scalability and performance of operations like smart contracts and potentially NFTs without impacting the network's transaction processing speed.

Virtual Voting

A mechanism for achieving consensus in a distributed system, which can be used in some forms of distributed ledger technology. Virtual Voting is one of two consensus components created for the Hedera (the other consensus component is gossip about gossip). These two components function dynamically to build a fair (linear) ordering of transactions without explicitly casting ballots.

Vyper

Vyper is an experimental, statically typed contract programming language meant to resemble Python. Like objects in OOP, each contract contains state variables, functions, and common data types. Contract-specific features include event notifiers for listeners, and custom global variables, and global constants. To learn more about Vyper, check out the documentation maintained by the Vyper team [here](#).

W

Web1

The first generation of the internet is primarily read-only and static information that links to each other. Think of a self-hosted, personal web page or a blog using a platform such as WordPress or Squarespace. You primarily share the information with others via URLs. Web1 is centralized or operated by companies that control servers that service customers.

Web2

The second generation of the internet, the current state of the internet, also referred to as the read-write web, is a web of social interactions linking users to each other. Think of social media sites such as Instagram and Twitter, where you share and consume algorithm-based data feeds managed by tech companies. You often share information within the platform by liking, sharing, re-posting with your own comments, and more. Though much of web2 content is created by users, it is not necessarily owned by them. Both web1 and web2 are centralized or operated by companies that control servers that serve customers.

Web3

The envisioned next generation of the internet is characterized by decentralization, distributed ledger technology, and real-time, persistent data, also referred to as the read-write-own web. Web3 is the Internet, plus added decentralization components such as DLTs & blockchains, cryptocurrencies, and NFTs. Whereas web2 is centralized or operated by companies that control servers that serve customers, web3 is decentralized, built, and operated by users.

Weibars

The EVM returns gas information in Weibars (introduced in HIP-410). One weibars is 10^{18} HBARs, which translates to 1 tinybar is 10^{10} weibars. As noted in HIP-410, this is to maximize compatibility with third-party tools that expect ether

units to be operated on in fractions of 10^{18} , also known as a Wei.

White Paper

In the DLT world, a white paper is a document containing technical information about a proposed project, such as an Initial Coin Offering (ICO). A white paper typically aims at potential investors and provides background information about the project, how the technology will work, what problems it may solve, and who would benefit from the solution. Bitcoin: A Peer-to-Peer Electronic Cash System, published by Satoshi Nakamoto in 2008, is a famous White Paper that "propose(d) a solution to the double-spending problem using a peer-to-peer network."

Z

Zk-SNARK

zk-SNARK is an acronym for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. In a ledger, zk-SNARK is a multi-theorem proof system that grants the power to control access to validated transaction information. This means that ledger data can be gated.

hello-future-hackathon.md:

Hello Future Hackathon

Are you ready to build the unimaginable on Hedera?

{% embed url="https://www.canva.com/design/DAGJ9iEWrU/ZFU25tkqBDsvnfRu-k4Fmw/view" %}

Join the Hello Future Hackathon and Innovate with Hedera!

From July 22 to August 20, enter a virtual playing field full of creativity and collaboration. Compete for a piece of a prize pool valued at up to \$300,000 and transform your skills into lasting change within the Web3 space.

Don't miss out, Register by August 18!

{% embed url="https://angelhack.typeform.com/hellofuturehack" %}

meetups.md:

description: >-

Hedera Hashgraph Meetups offer a great opportunity for community members to learn about how to use Hedera network services, best development practices, and much more.

cover: ../gitbook/assets/28ultraviolet.jpg

coverY: 0

Meetups

We currently have 54 Meetup groups worldwide, including one online group for virtual meetups. Check out our extensive Meetup list below and join a group that supports your needs. We are very excited to see you there!

Online

https://www.meetup.com/Hedera-Hashgraph-Virtual-Meetup/	Monthly Virtual Meetup	

Europe

https://www.meetup.com/Hashgraph-Paris-France/	Paris	https://www.meetup.com/Hashgraph-Paris-France/
https://www.meetup.com/Hashgraph-Berlin-Germany1/	Berlin , Germany	https://www.meetup.com/Hashgraph-Berlin-Germany1/
https://www.meetup.com/Hashgraph-Frankfurt-Germany/	Frankfurt , Germany	https://www.meetup.com/Hashgraph-Frankfurt-Germany/
https://www.meetup.com/Hashgraph-Athens-Greece1/	Athens , Greece	https://www.meetup.com/Hashgraph-Athens-Greece1/
https://www.meetup.com/Hashgraph-Ireland/	Dublin , Ireland	https://www.meetup.com/Hashgraph-Ireland/
https://www.meetup.com/Hedera-Hashgraph-Milan-Italy/	Milan , Italy	https://www.meetup.com/Hedera-Hashgraph-Milan-Italy/
https://www.meetup.com/Hashgraph-Vilnius-Lithuania/	Vilnius , Lithuania	https://www.meetup.com/Hashgraph-Vilnius-Lithuania/
https://www.meetup.com/Hedera-Hashgraph-Valletta-Malta/	Valletta , Malta	https://www.meetup.com/Hedera-Hashgraph-Valletta-Malta/
https://www.meetup.com/Hashgraph-Amsterdam-Netherlands/	Amsterdam , NL	https://www.meetup.com/Hashgraph-Amsterdam-Netherlands/
https://www.meetup.com/Hashgraph-Lisbon-Portugal/	Lisbon , Portugal	https://www.meetup.com/Hashgraph-Lisbon-Portugal/
https://www.meetup.com/Hashgraph-Stockholm-Sweden/	Stockholm , Sweden	https://www.meetup.com/Hashgraph-Stockholm-Sweden/
https://www.meetup.com/Hashgraph-Zurich-Switzerland/	Zürich , Switzerland	https://www.meetup.com/Hashgraph-Zurich-Switzerland/

align="center">London, UK GB</td><td align="center"></td><td>https://www.meetup.com/Hashgraph-London-United-Kingdom/</td></tr></tbody></table>

North Asia

<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Moscow, Russia RU</td><td></td><td>https://www.meetup.com/Hashgraph-Moscow-Rus/</td></tr><tr><td align="center">St. Petersburg, Russia RU</td><td></td><td>https://www.meetup.com/Hashgraph-Saint-Petersburg-Rus/</td></tr></tbody></table>

Africa

<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Cairo, Egypt EG</td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Cairo-Egypt/</td></tr><tr><td align="center">Nairobi, Kenya KE</td><td></td><td>https://www.meetup.com/Hashgraph-Nairobi-Kenya/</td></tr><tr><td align="center">Lagos, Nigeria NG</td><td></td><td>https://www.meetup.com/Hashgraph-Lagos-Nigeria/</td></tr><tr><td align="center">Riyadh, Saudi Arabia SA</td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Riyadh/</td></tr><tr><td align="center">Johannesburg, SA ZA</td><td></td><td>https://www.meetup.com/Hashgraph-Johannesburg-South-Africa/</td></tr></tbody></table>

Americas

Canada CA

<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Toronto, ON</td><td></td><td>https://www.meetup.com/Hashgraph-Toronto-Canada/</td></tr><tr><td align="center">Vancouver, BC</td><td></td><td>https://www.meetup.com/Hashgraph-Vancouver-Canada/</td></tr></tbody></table>

United States us

https://www.meetup.com/Hashgraph-Los-Angeles-California/ Los Angeles, CA</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Los-Angeles-California/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-San-Francisco-California/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-San-Francisco-California/</td></tr><tr><td align="center">https://www.meetup.com/hashgraph-denver-boulder-colorado/</td><td></td><td></td><td>https://www.meetup.com/hashgraph-denver-boulder-colorado/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Washington-DC/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Washington-DC/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Meetup-Atlanta/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Meetup-Atlanta/</td></tr><tr><td align="center">https://www.meetup.com/Hedera-Hashgraph-Savannah-Georgia/</td><td></td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Savannah-Georgia/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Chicago-Illinois/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Chicago-Illinois/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Boston-Massachusetts/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Boston-Massachusetts/</td></tr><tr><td align="center">https://www.meetup.com/Hedera-Hashgraph-Michigan/</td><td></td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Michigan/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-New-York-City-New-York/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-New-York-City-New-York/</td></tr><tr><td align="center">https://www.meetup.com/Hedera-Hashgraph-Charlotte-NC/</td><td></td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Charlotte-NC/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Portland-Oregon/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Portland-Oregon/</td></tr><tr><td align="center">https://www.meetup.com/hashgraph-philadelphia/</td><td></td><td></td><td>https://www.meetup.com/hashgraph-philadelphia/</td></tr><tr><td align="center">https://www.meetup.com/Hashgraph-Austin-Texas/</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Austin-Texas/</td></tr><tr><td align="center"><a href="https://www.meetup.com/Hashgraph-Dallas-

Texas/">Dallas, TX</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Dallas-Texas/</td></tr><tr><td align="center">Providence, RI</td><td></td><td></td><td>https://www.meetup.com/Hedera-Hashgraph-Providence-Rhode-Island/</td></tr><tr><td align="center">Seattle, WA</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Seattle-Washington/</td></tr></tbody></table>

South America

<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">São Paulo, Brazil BR</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Sao-Paulo-Brazil/</td></tr></tbody></table>

Asia & Pacific

<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center">Sydney, Australia AU</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Sydney-Australia/</td></tr><tr><td align="center">Beijing, China CN</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Beijing-China/</td></tr><tr><td align="center">Bangalore, India IN</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Bangalore-India/</td></tr><tr><td align="center">Delhi, India IN</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Delhi-India/</td></tr><tr><td align="center">Mumbai, India IN</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Mumbai-India/</td></tr><tr><td align="center">Thiruvananthapuram, IN</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Thiruvananthapuram-India/</td></tr><tr><td align="center">Tel Aviv, Israel IL</td><td></td><td></td><td>https://www.meetup.com/Hashgraph-Tel-Aviv-Israel/</td></tr><tr><td align="center">Tokyo, Japan JP</td><td></td><td></td><td>https://www.meetup.com/hashgraphjapan/</td></tr><tr><td align="center">al-Kuwayt, Kuwait KW</td><td></td><td></td><td>https://www.meetup.com/


```
visible: true
description:
  visible: false
tableOfContents:
  visible: true
outline:
  visible: true
pagination:
  visible: true
```

Support & Community

demo-applications.md:

```
cover: >-
  ../.gitbook/assets/Hero-Desktop-EnterpriseApplications2022-12-08-192047ivzd
  (2).webp
coverY: 0
---
```

Demo Applications

```
{% tabs %}
{% tab title="Java" %}
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th align="center"></th><th align="center"></th><th data-
hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><strong>Corda Notary</strong></td><td align="center">Developer:
Hedera</td><td align="center"><a href="https://github.com/hashgraph/corda-
notary-hedera"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hashgraph/corda-notary-hedera">https://github.com/
hashgraph/corda-notary-hedera</a></td></tr><tr><td align="center"><strong>HCS
Secure Extension Components</strong></td><td align="center">Developer:
Hedera</td><td align="center"><a href="https://github.com/hashgraph/hedera-hcs-
sxc-java"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hashgraph/hedera-hcs-sxc-java">https://github.com/
hashgraph/hedera-hcs-sxc-java</a></td></tr><tr><td align="center"><strong>Hedera
JSON-RPC Relay</strong></td><td align="center">Developer: Hedera</td><td
align="center"><a href="https://github.com/hashgraph/hedera-json-rpc-
relay"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hashgraph/hedera-json-rpc-relay">https://github.com/
hashgraph/hedera-json-rpc-relay</a></td></tr><tr><td align="center"><a
href="https://docs.hedera.com/hedera-transaction-tool-demo/"><strong>Hedera
Transaction Tool</strong></a></td><td align="center">Developer: Hedera</td><td
align="center"><a href="https://github.com/hashgraph/hedera-transaction-tool-
demo"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hashgraph/hedera-transaction-tool-demo">https://
github.com/hashgraph/hedera-transaction-tool-demo</a></td></tr><tr><td
align="center"><strong>Log4J Integration</strong></td><td
align="center">Developer: Hedera</td><td align="center"><a
href="https://github.com/hashgraph/log4j2-hedera"><strong>GITHUB</strong></a></
td><td><a href="https://github.com/hashgraph/log4j2-hedera">https://github.com/
hashgraph/log4j2-hedera</a></td></tr><tr><td align="center"><strong>Logstash
Integration</strong></td><td align="center">Developer: Hedera</td><td
align="center"><a href="https://github.com/hashgraph/logstash-output-
hedera"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hashgraph/logstash-output-hedera">https://github.com/
hashgraph/logstash-output-hedera</a></td></tr><tr><td
align="center"><strong>Proof-of-action Microservice</strong></td><td
align="center">Developer: Hedera</td><td align="center"><a
```


	https://github.com/hashgraph/MyHbarWallet
Non-Fungible Tokens	Developer: Hedera
https://github.com/hashgraph/hedera-hts-demo#nfts	https://github.com/hashgraph/hedera-hts-demo#nfts
Proof-Of-Action Pasteboard	Developer: Hedera
https://github.com/hashgraph/hedera-proof-of-action-demo-pasteboard	https://github.com/hashgraph/hedera-proof-of-action-demo-pasteboard
Theft Prevention	Developer: Hedera
https://github.com/hashgraph/hedera-theft-prevention-demo	https://github.com/hashgraph/hedera-theft-prevention-demo

More coming soon!

Go
Audit Logs
Developer: AdsDax
https://github.com/hashgraph/hello-hedera-audit-log-go
Hyperledger Fabric Plugin
Developer: Hyperledger
https://github.com/hashgraph/fabric-samples-hcs/tree/feature/hcs

More coming soon!

Coming soon!

<details>

<summary>Do you have a demo application to add to this list? Or want to request others to get built?</summary>

Please refer to the contributing guide and open an issue in the hedera-docs repository and include the following information within the issue:

Demo application name
Developer/maintainer name
Link to the demo application GitHub repository

</details>

getting-started-javascript.md:

Getting Started: JavaScript

Background

Hedera is the public ledger built on the lightning fast hashgraph consensus algorithm. You can use Hedera like you would a blockchain; send cryptocurrency, run smart contracts, even store files!

We're getting close to availability of Hedera's JavaScript SDK, which will make it even easier to build applications. In this post I'll show you how you can get your environment setup and start using Hedera Hashgraph with Node.js, one of the most popular environments in the world.

Step 1: Create an Account

In order to use the Hedera Public Testnet you'll need an Account. You can get one by signing up on portal.hedera.com, or maybe a friend who already is on the public testnet can create one for you.

Step 2: Set up node.js environment

In this simple example, we'll create the bare minimum node.js environment we're going to need.

If you already have a node environment set up that you'd like to use, skip to step 3.

> Note: The following steps assume you're working in the mac terminal

2.1. Create a new directory for our example & move into it.

```
mkdir hello-hedera-js-sdk && cd hello-hedera-js-sdk
```

2.2. Initialize a node.js project in this new directory.

```
npm init
```

Note: you can just say "yes" to all of the defaults and/or plugin what makes sense. It's an example!

Here's mine for reference.

```
javascript
{
  "name": "hello-hedera-js-sdk",
  "version": "1.1.12",
  "description": "A hello world project for the Hedera Hashgraph JavaScript SDK",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Your Name",
  "license": "Apache-2.0"
}
```

2.3. Switch environments, and open your directory in a suitable text editor of your choosing.

I personally really like VS Code if you haven't checked it out recently!

2.4. Create your .env file in the root of your directory.

Now you can add the following information, provided from your Hedera Portal Account.

```
yaml
Operator ID and Key
OPERATORID=YOUR-TESTNET-ACCOUNT-ID
OPERATORKEY=YOUR-TESTNET-PRIVATE-KEY
```

> Don't have an account yet? Sign up here.

2.4. Create an index.js file in the 'root' of your directory.

You can just add this to the file, so we can make sure you have node configured properly.

```
console.log("hello node.js!");
```

2.5. Test out your node.js installation.

Switch environments back over to your terminal.

You should be able to run node -v to get your current version.

Presuming you're all setup with node, running node index.js should output hello node.js!

If you don't get an appropriate response, you may need to install node.

Step 3: Install the Hedera Hashgraph JS SDK

Now that you have your node environment setup, we can get started with Hedera's JS SDK!

> View Hedera Hashgraph's JavaScript SDK here

Install it with your favorite package manager.

```
text
// install Hedera's JS SDK with NPM
// This example uses Hedera JavaScript SDK v1.1.12
npm install --save @hashgraph/sdk

// Install with Yarn
yarn add @hashgraph/sdk
```

You'll also likely want to install dotenv with your favorite package manager.

This will allow our node environment to use your keys & Account ID we saved earlier.

```
text
// install with NPM
npm install dotenv

// Install with Yarn
yarn add dotenv
```

Step 4: Finally, the fun part

Now we will go through some simple examples you can do!

3.1. Example 1: Checking your balance

Update your index.js with the following examples.

Note: there are inline comments to help you understand what's going on!

```
javascript
// Import the Hedera Hashgraph JS SDK
// Example uses Hedera JavaScript SDK v1.1.12
const { Client, AccountBalanceQuery } = require("@hashgraph/sdk");
// Allow access to our .env file variables
require("dotenv").config();

// Grab your account ID and private key from the .env file
const operatorAccountId = process.env.OPERATORID;
const operatorPrivateKey = process.env.OPERATORKEY;

// If we weren't able to grab it, we should throw a new error
if (operatorPrivateKey == null ||
    operatorAccountId == null ) {
    throw new Error("environment variables OPERATORKEY and OPERATORID must be
present");
}

// Create our connection to the Hedera network
// The Hedera JS SDK makes this really easy!
const client = Client.forTestnet();

// Set your client account ID and private key used to pay for transaction fees
and sign transactions
client.setOperator(operatorAccountId, operatorPrivateKey);

// Hedera is an asynchronous environment :)
(async function() {

    // Attempt to get and display the balance of our account
    var currentBalance = await new
AccountBalanceQuery().setAccountId(operatorAccountId).execute(client);
    console.log("account balance:", currentBalance);
})();
```

Copy and paste this into your index.js, and if everything is setup successfully, we can now run it!

Switch back to your terminal and run node index.js again.

If successful, within a few seconds we should see something like account balance: 100500005000

> Congratulations! You've submitted your first query with Hedera's JS SDK.

3.2. Example 2: Transferring hbar

```
javascript
// Import the Hedera Hashgraph JS SDK
// Example uses Hedera JavaScript SDK v1.1.12
const { Client, CryptoTransferTransaction, AccountId } =
require("@hashgraph/sdk");
// Allow access to our .env file variables
require("dotenv").config();
```

```

// Grab your account ID and private key from the .env file
const operatorAccountId = process.env.OPERATORID;
const operatorPrivateKey = process.env.OPERATORKEY;

// If we weren't able to grab it, we should throw a new error
if (operatorPrivateKey == null ||
    operatorAccountId == null ) {
    throw new Error("environment variables OPERATORKEY and OPERATORID must be
present");
}

// Create our connection to the Hedera network
// The Hedera JS SDK makes this really easy!
const client = Client.forTestnet();

// Set your client default account ID and private key used to pay for
transaction fees and sign transactions
client.setOperator(operatorAccountId, operatorPrivateKey);

// Hedera is an asynchronous environment :)
(async function() {
    console.log("balance before transfer:", (await
client.getAccountBalance(operatorAccountId)));

    const receipt = await (await new CryptoTransferTransaction()
        .addSender(operatorAccountId, 1)
        .addRecipient("0.0.3", 1)
        .setTransactionMemo("sdk example")
        .execute(client))
        .getReceipt(client);

    console.log(receipt);
    console.log("balance after transfer:", (await
client.getAccountBalance(operatorAccountId)));

})();

```

> Congratulations! You've submitted your first transaction with Hedera's JS SDK.

This is the end of my brief getting started example. You have now:

- Created a Hedera Testnet Account
- Setup the Hedera JS SDK in a node environment
- Submit your first transaction and queries!

Want to keep learning about Hedera development? Here's some resources.

- Documentation
 - JS SDK examples
- Media
 - Install the JS SDK
 - Environment set-up

Is there anything else you'd like a tutorial about? Let me know on twitter.
{% hint style="info" %}
Have a question?
Ask it on StackOverflow
{% endhint %}

hcs-submit-your-first-message.md:

HCS: Submit your first message!

Background

With the Hedera Consensus Service you can develop applications like stock markets, audit logs, stable coins, or new network services that require high throughput and decentralized trust. This is made possible by having direct access to the native speed, security, and fair ordering guarantees of the hashgraph consensus algorithm, with the full trust of the Hedera ledger.

Components of the Hedera Consensus Service/Terminology

Hedera client - a Hedera client sends transactions to a Hedera network node for consensus. The corresponding transaction types for the Hedera Consensus Service include create a topic, update a topic, submit messages, delete a topic, and get the info for a topic.

Hedera network node - receives transactions from a client and submits it to the Hedera network for consensus

Mirror node client - a mirror node client is used to subscribe to a topic and get messages that are in consensus order from a mirror node.

Mirror node: Mirror nodes receive information from Hedera network consensus nodes, but do not participate in consensus themselves. You can get more information about mirror nodes [here](#).

Topic - a topic is the subject of information you would like to send messages to and what clients would subscribe to

Message - a message is the content published to the Hedera network

to a topic which gets placed in consensus order

Subscriber - a client that subscribes to a desired topic in order to receive the appropriate messages

Publisher - publishes messages to a topic

HCS Flow

Create a topic by submitting a transaction from the Hedera client to a Hedera network node → Mirror node client subscribes to the topic from mirror node → Publish a message to a topic by submitting a transaction from the Hedera client to a Hedera network node for consensus → Mirror node client receives messages published to the topic from the mirror node

Getting Started with Hedera Consensus Service \ (Testnet\)

What you will need to be successful with this tutorial:

- ✳ Testnet account ID and associated private key
- ✳ Hedera Java SDK
- ✳ Mirror node host:port
- ✳ IDE of your choice

1. Get a testnet account

You will need a testnet account ID and private key to use HCS. If you do not already have one, visit the Hedera portal to create your profile and receive your testnet account ID. A friend already on testnet could also generously create one for you.

2. Get access to a mirror node

You will need a mirror node's host:port information so that you can subscribe to topics and receive the associated messages. If you do not have access to a mirror node, you can use the mirror node endpoints managed by Hedera [here](#).

3. Create a new maven project in your favorite IDE

Add the following dependencies to your pom.xml file

```
java
<dependency>
  <groupId>com.hedera.hashgraph</groupId>
  <artifactId>sdk</artifactId>
  <version>1.1.3</version>
</dependency>

<!-- SELECT ONE: -->
<!-- netty transport (for server or desktop applications) -->
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-netty-shaded</artifactId>
  <version>1.24.0</version>
</dependency>
<!-- netty transport, unshaded (if you have a matching Netty dependency already)
-->
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-netty</artifactId>
  <version>1.24.0</version>
</dependency>
<!-- okhttp transport (for lighter-weight applications or Android) -->
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-okhttp</artifactId>
  <version>1.24.0</version>
</dependency>
<dependency>
  <groupId>io.github.cdimascio</groupId>
  <artifactId>java-dotenv</artifactId>
  <version>5.1.3</version>
</dependency>
```

4. Set-up your environment variables

Create a .env file in the projects root directory

Add the following:

OPERATORID: Your testnet account ID goes here

OPERATORKEY: Your testnet account ID private key goes here

Add the following:

MIRRORNODEADDRESS: Insert the mirror node host:port information here

Your environment set-up is now complete

Sample .env file:

```
text
Operator ID and Key
OPERATORID=
OPERATORKEY=
Mirror Node Address
MIRRORNODEADDRESS=
```

5. Create a new HCS class

Create a new class and title it something like HederaConsensusService.java

6. Connect to the Hedera testnet

Here we are going to connect to the Hedera testnet and set the operator information with your testnet account ID and private key. The operator is responsible to pay transaction fees and sign all transactions that will be generated in this tutorial. Luckily, this is testnet so you will have unlimited hbars to use in this development environment!

```
java
// Grab the OPERATORID and OPERATORKEY from the .env file
private static final AccountId OPERATORID =
AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATORID")));
private static final Ed25519PrivateKey OPERATORKEY =
Ed25519PrivateKey.fromString(Objects.requireNonNull(Dotenv.load().get("OPERATORKEY")));

// Build Hedera testnet client
Client client = Client.forTestnet();

// Set the operator account ID and operator private key
client.setOperator(OPERATORID, OPERATORKEY);
```

7. Connect to a Hedera mirror node

```
java
// Grab the mirror node endpoint from the .env file
private static final String MIRRORNODEADDRESS =
Objects.requireNonNull(Dotenv.load().get("MIRRORNODEADDRESS"));

// Build the mirror node client
final MirrorClient mirrorClient = new MirrorClient(MIRRORNODEADDRESS);
```

8. Create your first topic!

To create your first topic, we will use the `CreateTopicTransaction` constructor, set its properties, and submit it to the Hedera network. You will want to grab the topic ID so later you can subscribe to that topic via the mirror node client. Topic IDs are in the following format: 0.0.10

```
java
//Create a new topic
final TransactionId transactionId = new ConsensusTopicCreateTransaction()
    .execute(client);

//Grab the newly generated topic ID
final ConsensusTopicId topicId =
transactionId.getReceipt(client).getConsensusTopicId();

System.out.println("Your topic ID is: " +topicId);
```

9. Subscribe to a topic

Now we will shift our attention to the mirror node client and subscribe to the topic created. We will achieve this by referencing the topic's `topicId`. We will also print the consensus timestamp and message to the console.

```
java
new MirrorConsensusTopicQuery()
    .setTopicId(topicId)
    .subscribe(mirrorClient, resp -> {
        String messageAsString = new String(resp.message,
```

```
StandardCharsets.UTF8);
```

```
        System.out.println(resp.consensusTimestamp + " received topic message: "
+ messageAsString);
    },
    // On gRPC error, print the stack trace
    Throwable::printStackTrace);
```

10. Submit a message to a topic

Now that we have created a topic and subscribed to that topic, we are ready to submit a message using the `ConsensusSubmitTransaction` constructor and submit to the Hedera network

The below example will submit a message with the message as "hello, HCS!"

```
java
//Submit a message to a topic
new ConsensusMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("hello, HCS! ")
    .execute(client)
    .getReceipt(client);
```

If you have successfully followed the steps in this tutorial, you should see the following print to your console 🖨 :

```
2020-01-17T09:01:03.990648Z received topic message: hello, HCS!
```

NOTE: It may take 10-15 seconds before the message appears on your console from the mirror node.

Having trouble or have any comments, suggestions, or feedback?

Connect with us on Discord🗨!

Have a question?

Ask it on StackOverflow

README.md:

cover: >-

../.gitbook/assets/Cat-Hero-12-hero@2x-100Exchanges2022-12-07-020913ugkr.webp

coverY: -751.6690909090909

layout:

cover:

visible: true

size: full

title:

visible: true

description:

visible: false

tableOfContents:

visible: true

outline:

visible: true

pagination:

visible: true

Tutorials

starter-projects.md:

cover: ../.gitbook/assets/Hero-Desktop-Tooling2022-12-07-021130ayix (1) (1).webp
coverY: -69

Starter Projects

```
{% tabs %}
{% tab title="Java" %}
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th align="center"></th><th align="center"></th><th data-
hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><strong>Aviator Framework</strong></td><td
align="center">Maintainer: Community</td><td align="center"><a
href="https://github.com/TxMQ/aviator-core"><strong>GITHUB</strong></a></
td><td><a href="https://github.com/TxMQ/aviator-core">https://github.com/TxMQ/
aviator-core</a></td></tr><tr><td align="center"><strong>Spring
Framework</strong></td><td align="center">Maintainer: Community</td><td
align="center"><a href="https://github.com/rahul-kothari/hedera-starter-
spring"><strong>GITHUB</strong></a></td><td><a href="https://github.com/rahul-
kothari/hedera-starter-spring">https://github.com/rahul-kothari/hedera-starter-
spring</a></td></tr></tbody></table>
```

More coming soon!

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th align="center"></th><th align="center"></th><th data-
hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><strong>React.js</strong></td><td align="center">Maintainer:
Community</td><td align="center"><a href="https://github.com/hedera-dev/cra-
hedera-dapp-template"><strong>GITHUB</strong></a></td><td><a
href="https://github.com/hedera-dev/cra-hedera-dapp-template">https://
github.com/hedera-dev/cra-hedera-dapp-template</a></td></tr><tr><td
align="center"><strong>Express.js (chat)</strong></td><td
align="center">Maintainer: Hedera</td><td align="center"><a
href="https://github.com/hashgraph/hedera-hcs-chat-js"><strong>GITHUB</
strong></a></td><td><a href="https://github.com/hashgraph/hedera-hcs-chat-
js">https://github.com/hashgraph/hedera-hcs-chat-js</a></td></tr><tr><td
align="center"><strong>Express.js (Nuxt)</strong></td><td
align="center">Maintainer: Community</td><td align="center"><a
href="https://github.com/scalemaildev/hashgraphnuxtchat"><strong>GITHUB</
strong></a></td><td><a href="https://github.com/scalemaildev/hashgraphnuxtchat">https://github.com/
scalemaildev/hashgraphnuxtchat</a></td></tr></tbody></table>
```

More coming soon!

```
{% endtab %}
```

```
{% tab title="Go" %}
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th align="center"></th><th align="center"></th><th data-
hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><strong>Hyperledger Fabric</strong></td><td
align="center">Maintainer: Hedera</td><td align="center"><a href="broken-
reference/"><strong>REFERENCE</strong></a></td><td><a href="broken-
reference/">broken-reference</a></td></tr></tbody></table>
```

More coming soon!

```
{% endtab %}  
{% endtabs %}
```

<details>

<summary>Do you have a starter project to add to this list? Or want to request others to get built?</summary>

Please refer to the contributing guide and open an issue in the hedera-docs repository and include the following information within the issue:

Starter project framework name
Developer/maintainer name
Link to the GitHub repository

</details>

query-messages-with-mirror-node.md:

Query Messages with Mirror Node

Summary

In the first tutorial, "Submit Your First Message," you have learned how to submit a message to a topic.

In this tutorial, you will learn how to query the Hedera Mirror Node API to retrieve and filter messages.

Prerequisites

We recommend that you complete the "Submit Your First Message" tutorial here to get a basic understanding of the Hedera Consensus Service. This example does not build upon the previous examples.

Table of Contents

1. Create Topic and Submit Messages
2. Query Hedera Mirror Node API
3. Retrieve Message
4. Advanced Filtering

1. Create a topic and submit three messages

For this tutorial, create a new topic and submit three messages to this topic on testnet. You will use the retrieved topic ID to query for messages via the Hedera Mirror Node API.

Copy and execute the following code. Make sure to write down your topic ID. The topic ID will be in 0.0.topicId format (ex: 0.0.1234).

```
{% tabs %}  
{% tab title="Java" %}  
java  
// Create a new topic  
TransactionResponse txResponse = new TopicCreateTransaction()  
    .setSubmitKey(myPrivateKey.getPublicKey())
```

```

        .execute(client);

// Get the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

// Get the topic ID
TopicId topicId = receipt.topicId;

// Log the topic ID
System.out.println("Your topic ID is: " +topicId);

// Submit messages
TransactionResponse submitMessage1 = new TopicMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("Message 1")
    .execute(client);

TransactionResponse submitMessage2 = new TopicMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("Message 2")
    .execute(client);

TransactionResponse submitMessage3 = new TopicMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("Message 3")
    .execute(client);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create a new public topic
let txResponse = await new TopicCreateTransaction().execute(client);

// Grab the newly generated topic ID
let receipt = await txResponse.getReceipt(client);
let topicId = receipt.topicId;
console.log(Your topic ID is: ${topicId});

// Submit messages
await new TopicMessageSubmitTransaction({
    topicId: topicId,
    message: "Message 1",
}).execute(client);

await new TopicMessageSubmitTransaction({
    topicId: topicId,
    message: "Message 2",
}).execute(client);

await new TopicMessageSubmitTransaction({
    topicId: topicId,
    message: "Message 3",
}).execute(client);

{% endtab %}

{% tab title="Go" %}
go
// Create a new topic
transactionResponse, err := hedera.NewTopicCreateTransaction().
    SetSubmitKey(myPrivateKey.PublicKey()).
    Execute(client)

```

```

if err != nil {
    println(err.Error(), ": error creating topic")
    return
}

// Get the topic create transaction receipt
transactionReceipt, err := transactionResponse.GetReceipt(client)

if err != nil {
    println(err.Error(), ": error getting topic create receipt")
    return
}

// Get the topic ID from the transaction receipt
topicID := transactionReceipt.TopicID

// Log the topic ID to the console
fmt.Printf("Your topic ID is: %v\n", topicID)

// Submit messages
submitMessage1, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Message 1")).
    SetTopicID(topicID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

submitMessage2, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Message 2")).
    SetTopicID(topicID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

submitMessage3, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Message 3")).
    SetTopicID(topicID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

{% endtab %}
{% endtabs %}

```

The output in your console should look like this after executing the above setup code:

```
Your topic ID is: 0.0.<4603900>
```

Next, let's query the mirror node to retrieve data.

2. Query the Hedera Mirror Node API

Now all three messages have been submitted to your topic ID on testnet, let's query the mirror node. Let's use the testnet endpoint for the Hedera Mirror Node API to query for all messages for your topic ID. Make sure to replace the topic ID with the topic ID you've written down and execute the request in your browser or tool of choice.

```
<pre><code><strong>// Replace &#x3C;topicId>
</strong><strong>https://testnet.mirrornode.hedera.com/api/v1/topics/
&#x3C;topicID>/messages
</strong>
// Example
https://testnet.mirrornode.hedera.com/api/v1/topics/0.0.4603900/messages
</code></pre>
```

The result should look similar to the API result below, with three messages being returned. The actual message contents are base64 encoded. If you want to verify the message contents, you can use this decoder website or decode it using code yourself.

<details>

<summary>✔ API result</summary>

```
json
{
  "links": { "next": null },
  "messages": [
    {
      "consensustimestamp": "1683553059.977315003",
      "message": "TWVzc2FnZSAx",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"vKnW8bYSjhYtHtMN0Jwrkfyv77kKc4EREDycKE2L37aU0SYLx+g6HB9ah7CTFSxl",
      "runninghashversion": 3,
      "sequencenumber": 1,
      "topicid": "0.0.4603900"
    },
    {
      "consensustimestamp": "1683553060.092158003",
      "message": "TWVzc2FnZSAy",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"JM8LpmRwUVc+wiwzRHBgcCv8uJkbfan8BV2QPC0hBMgXA9KIvy8Tw0oXsukgmeC+",
      "runninghashversion": 3,
      "sequencenumber": 2,
      "topicid": "0.0.4603900"
    },
    {
      "consensustimestamp": "1683553060.328178003",
      "message": "TWVzc2FnZSAz",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"OL6EUxAayPsRiuhwflX5heRgIGEwmHcJU7bs2qBPeJNo0iMQmaY0H6G/pXJ7wGQz",
      "runninghashversion": 3,
      "sequencenumber": 3,
      "topicid": "0.0.4603900"
    }
  ]
}
```

</details>

3. Retrieve a specific message by sequence number

In this section, you'll learn how to query messages by a sequence number. Each message you submit to a topic receives a sequence number starting from 1.

If you take a look at the REST API docs for Topics, you'll find the first query `/api/v1/topics/{topicId}/messages`. If you expand this section, you'll find all query parameters.

<figure><figcaption><p>Query parameters REST API</p></figcaption></figure>

Execute the below request to retrieve the message with sequence number 2.

```
// Replace <topicId>
https://testnet.mirrornode.hedera.com/api/v1/topics/<topicID>/messages?
sequencenumber=2
```

```
// Example
https://testnet.mirrornode.hedera.com/api/v1/topics/0.0.4603900/messages?
sequencenumber=2
```

<details>

<summary>✔ API result</summary>

Only message two is returned by the Hedera Mirror Node.

```
json
{
  "links": { "next": null },
  "messages": [
    {
      "consensustimestamp": "1683553060.092158003",
      "message": "TWVzc2FnZSAy",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"JM8LpmRwUVc+wiwzRHBgcCv8uJkbfan8BV2QPC0hBMgXA9KIvy8Tw0oXsukgmeC+",
      "runninghashversion": 3,
      "sequencenumber": 2,
      "topicid": "0.0.4603900"
    }
  ]
}
```

</details>

4. Advanced filtering methods for HCS messages

This section explores advanced filtering methods using query modifiers. The OpenAPI specification for the Hedera Mirror Node REST API shows all details for query parameters (e.g. `timestampQueryParam`).

Possible query modifiers are:

Greater than (gt) / greater than or equal (gte)
Lower than (lt) / lower than or equal (lte)
Equal to (eq)
Not equal to (ne)


You can use these modifiers for query parameters like sequencenumber and timestamp (consensus timestamp).

In this step, let's query all messages with a sequence number greater than or equal to 2. To do so, let's use the gte query parameter modifier and assign it the value 2, like this: sequencenumber=gte:2.

```
// Replace <topicId>
https://testnet.mirrornode.hedera.com/api/v1/topics/<topicID>/messages?
sequencenumber=gte:2
```

```
// Example
https://testnet.mirrornode.hedera.com/api/v1/topics/0.0.4603900/messages?
sequencenumber=gte:2
```

<details>

<summary>  API result </summary>

Only message two is returned by the Hedera Mirror Node.

```
json
{
  "links": { "next": null },
  "messages": [
    {
      "consensustimestamp": "1683553060.092158003",
      "message": "TWVzc2FnZSAy",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"JM8LpmRwUVc+wiwzRHBgcCv8uJkbfan8BV2QPC0hBMgXA9KIvy8Tw0oXsukgmeC+",
      "runninghashversion": 3,
      "sequencenumber": 2,
      "topicid": "0.0.4603900"
    },
    {
      "consensustimestamp": "1683553060.328178003",
      "message": "TWVzc2FnZSAz",
      "payeraccountid": "0.0.2617920",
      "runninghash":
"OL6EUxAayPsRiuhwflX5heRgIGEWmHcJU7bS2qBPeyJNo0iMQmaY0H6G/pXJ7wGQz",
      "runninghashversion": 3,
      "sequencenumber": 3,
      "topicid": "0.0.4603900"
    }
  ]
}
```

</details>

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

<table data-card-size="large" data-view="cards"><thead><tr><th>

align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Michiel, Developer Advocate</p><p>GitHub LinkedIn</p></td><td align="center"><p>Editor: Simi, Sr. Software Manager</p><p>GitHub LinkedIn</p></td></tr><tr><td align="center"><p>https://www.linkedin.com/in/michielmulders/</td><td align="center"><p>https://www.linkedin.com/in/shunjan</td></tr></tbody></table>

README.md:

Consensus Service

submit-message-to-private-topic.md:

Submit Message to Private Topic

Summary

In the previous tutorial, "Submit Your First Message," you have learned how to submit a message to a public topic. It means anyone can send a message to the topic you created because you didn't set a Submit Key.

When setting a Submit Key, your topic becomes a private topic because each message needs to be signed by the Submit Key. Therefore, you can control who can submit messages to your topic. Of course, the data is still public, as is all data on a public ledger, but we say the topic is private because the topic is restricted by who can submit messages to it.

Prerequisites

We recommend you complete the "Submit Your First Message" tutorial here to get a basic understanding of the Hedera Consensus Service. This example does not build upon the previous examples.

✔ You can find a full code check for this tutorial at the bottom of this page.

Table of Contents

1. Create Private Topic
2. Subscribe to Topic
3. Submit Message
4. Code Check

1. Create a private topic

To create a private topic, you will use `setSubmitKey()` to set a Submit Key. This key needs to sign all messages someone sends to the topic. A message will be rejected if you don't sign the message or sign with an incorrect key. The cost of creating a private topic is the same as a public topic: \$0.01.

```

{% tabs %}
{% tab title="Java" %}
java
// Create a new topic
TransactionResponse txResponse = new TopicCreateTransaction()
    .setSubmitKey(myPrivateKey.getPublicKey())
    .execute(client);

// Get the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

// Get the topic ID
TopicId topicId = receipt.topicId;

// Log the topic ID
System.out.println("Your topic ID is: " +topicId);

// Wait 5 seconds between consensus topic creation and subscription creation
Thread.sleep(5000);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create a new topic
let txResponse = await new TopicCreateTransaction()
    .setSubmitKey(myPrivateKey.publicKey)
    .execute(client);

// Grab the newly generated topic ID
let receipt = await txResponse.getReceipt(client);
console.log(Your topic ID is: ${receipt.topicId});

// Wait 5 seconds between consensus topic creation and subscription creation
await new Promise((resolve) => setTimeout(resolve, 5000));

{% endtab %}

{% tab title="Go" %}
go
// Create a new topic
transactionResponse, err := hedera.NewTopicCreateTransaction().
    SetSubmitKey(myPrivateKey.PublicKey()).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating topic")
    return
}

// Get the topic create transaction receipt
transactionReceipt, err := transactionResponse.GetReceipt(client)

if err != nil {
    println(err.Error(), ": error getting topic create receipt")
    return
}

// Get the topic ID from the transaction receipt
topicID := transactionReceipt.TopicID

//Log the topic ID to the console
fmt.Printf("topicID: %v\n", topicID)

```

```
{% endtab %}
{% endtabs %}
```

2. Subscribe to a topic

The code used to subscribe to a public or private topic doesn't change. Anyone can listen to the messages you send to your private topic. You need to provide the `TopicMessageQuery()` with your topic ID to subscribe to it.

```
{% tabs %}
{% tab title="Java" %}
java
// Subscribe to the topic
new TopicMessageQuery()
    .setTopicId(topicId)
    .subscribe(client, resp -> {
        String messageAsString = new String(resp.contents,
StandardCharsets.UTF8);
        System.out.println(resp.consensusTimestamp + " received topic
message: " + messageAsString);
    });
{% endtab %}

{% tab title="JavaScript" %}
javascript
// Subscribe to the topic
new TopicMessageQuery()
    .setTopicId(topicId)
    .subscribe(client, null, (message) => {
        let messageAsString = Buffer.from(message.contents, "utf8").toString();
        console.log(
            `${message.consensusTimestamp.toString()} Received: ${messageAsString}
        );
    });
{% endtab %}

{% tab title="Go" %}
go
// Subscribe to the topic
, err = hedera.NewTopicMessageQuery().
    SetTopicID(topicID).
    Subscribe(client, func(message hedera.TopicMessage) {
        fmt.Println(message.ConsensusTimestamp.String(), "received topic
message ", string(message.Contents), "\r")
    })
{% endtab %}
{% endtabs %}
```

3. Submit a message

Now you are ready to submit a message to your private topic. To do this, you will use `TopicMessageSubmitTransaction()`. However, you need to sign this transaction with your Submit Key. The cost for sending a message to a private topic is the same as a public topic: \$0.0001.

```

{% tabs %}
{% tab title="Java" %}
java
// Send message to private topic
TransactionResponse submitMessage = new TopicMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("Submitkey set!")
    .freezeWith(client)
    .sign(myPrivateKey)
    .execute(client)

// Get the receipt of the transaction
TransactionReceipt receipt2 = submitMessage.getReceipt(client);

// Prevent the main thread from exiting so the topic message can be returned and
// printed to the console
Thread.sleep(30000);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Send message to private topic
let submitMsgTx = await new TopicMessageSubmitTransaction({
    topicId: topicId,
    message: "Submitkey set!",
})
    .freezeWith(client)
    .sign(myPrivateKey);

let submitMsgTxSubmit = await submitMsgTx.execute(client);

// Get the receipt of the transaction
let getReceipt = await submitMsgTxSubmit.getReceipt(client);

// Get the status of the transaction
const transactionStatus = getReceipt.status;
console.log("The message transaction status " + transactionStatus.toString());

{% endtab %}

{% tab title="Go" %}
go
// Prepare message to send to private topic
submitMessageTx, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Submitkey set!")).
    SetTopicID(topicID).
    FreezeWith(client)

if err != nil {
    println(err.Error(), ": error freezing topic message submit transaction")
    return
}

// Sign message with submit key
submitMessageTx.Sign(myPrivateKey)

// Submit message
submitTxResponse, err := submitMessageTx.Execute(client)
if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

```


```
// Get the receipt of the transaction
receipt, err := submitTxResponse.GetReceipt(client)

// Get the transaction status
transactionStatus := receipt.Status
fmt.Println("The message transaction status " + transactionStatus.String())

// Prevent the program from exiting to display the message from the mirror node
to the console
time.Sleep(30000)

{% endtab %}
{% endtabs %}

To conclude: The total cost to create a topic and send a message to it is
$0.0101.
```

Code Check 

<details>

<summary>Java</summary>

```
<pre class="language-java"><code class="lang-java">import
com.hedera.hashgraph.sdk.;
import io.github.cdimascio.dotenv.Dotenv;

import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeoutException;

public class CreateTopicTutorial {
    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException, InterruptedException {

        // Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        // Build your Hedera client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Create a new topic
        TransactionResponse txResponse = new TopicCreateTransaction()
            .setSubmitKey(myPrivateKey.getPublicKey())
            .execute(client);

        // Get the receipt
        TransactionReceipt receipt = txResponse.getReceipt(client);

        // Get the topic ID
        TopicId topicId = receipt.topicId;

        // Log the topic ID
        System.out.println("Your topic ID is: " +topicId);

        // Wait 5 seconds between consensus topic creation and subscription
creation
        Thread.sleep(5000);
```



```

        // Subscribe to the topic
        new TopicMessageQuery()
            .setTopicId(topicId)
            .subscribe(client, resp -> {
                String messageAsString = new String(resp.contents,
StandardCharsets.UTF8);
                System.out.println(resp.consensusTimestamp + " received
topic message: " + messageAsString);
            });

        // Send message to private topic
<strong>        TransactionResponse submitMessage = new
TopicMessageSubmitTransaction()
</strong>            .setTopicId(topicId)
                .setMessage("Submitkey set!")
                .freezeWith(client)
                .sign(myPrivateKey)
                .execute(client)

        // Get the receipt of the transaction
        TransactionReceipt receipt2 = submitMessage.getReceipt(client);

        // Wait before the main thread exits to return the topic message to the
console
        Thread.sleep(30000);
    }
}
</code></pre>

```

</details>

<details>

<summary>JavaScript</summary>

```

javascript
console.clear();
require("dotenv").config();
const {
    AccountId,
    PrivateKey,
    Client,
    TopicCreateTransaction,
    TopicMessageQuery,
    TopicMessageSubmitTransaction,
} = require("@hashgraph/sdk");

// Grab the OPERATORID and OPERATORKEY from the .env file
const myAccountId = process.env.MYACCOUNTID;
const myPrivateKey = process.env.MYPRIVATEKEY;

// Build Hedera testnet and mirror node client
const client = Client.forTestnet();

// Set the operator account ID and operator private key
client.setOperator(myAccountId, myPrivateKey);

async function submitPrivateMessage() {
    // Create a new topic
    let txResponse = await new TopicCreateTransaction()
        .setSubmitKey(myPrivateKey.publicKey)
        .execute(client);

    // Grab the newly generated topic ID

```

```

let receipt = await txResponse.getReceipt(client);
let topicId = receipt.topicId;
console.log(Your topic ID is: ${topicId});

// Wait 5 seconds between consensus topic creation and subscription creation
await new Promise((resolve) => setTimeout(resolve, 5000));

// Create the topic
new TopicMessageQuery()
  .setTopicId(topicId)
  .subscribe(client, null, (message) => {
    let messageAsString = Buffer.from(message.contents, "utf8").toString();
    console.log(
      ${message.consensusTimestamp.toString()} Received: ${messageAsString}
    );
  });

// Send message to private topic
let submitMsgTx = await new TopicMessageSubmitTransaction({
  topicId: topicId,
  message: "Submitkey set!",
})
.freezeWith(client)
.sign(myPrivateKey);

let submitMsgTxSubmit = await submitMsgTx.execute(client);
let getReceipt = await submitMsgTxSubmit.getReceipt(client);

// Get the status of the transaction
const transactionStatus = getReceipt.status;
console.log("The message transaction status: " +
transactionStatus.toString());
}

submitPrivateMessage();

```

</details>

<details>

<summary>Go</summary>

```

go
package main

import (
    "fmt"
    "os"
    "time"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {

    // Loads the .env file and throws an error if it cannot load the variables
    from that file corectly
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load enviroment variables from .env
file. Error:\n%\v\n", err))
    }

```

```

// Grab your testnet account ID and private key from the .env file
myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
if err != nil {
    panic(err)
}

myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
if err != nil {
    panic(err)
}

// Create your testnet client
client := hedera.ClientForTestnet()
client.SetOperator(myAccountId, myPrivateKey)

// Create a new topic
transactionResponse, err := hedera.NewTopicCreateTransaction().
    SetSubmitKey(myPrivateKey.PublicKey()).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating topic")
    return
}

// Get the topic create transaction receipt
transactionReceipt, err := transactionResponse.GetReceipt(client)

if err != nil {
    println(err.Error(), ": error getting topic create receipt")
    return
}

// Get the topic ID from the transaction receipt
topicID := transactionReceipt.TopicID

// Log the topic ID to the console
fmt.Printf("topicID: %v\n", topicID)

//Create the query to subscribe to a topic
, err = hedera.NewTopicMessageQuery().
    SetTopicID(topicID).
    Subscribe(client, func(message hedera.TopicMessage) {
        fmt.Println(message.ConsensusTimestamp.String(), "received
topic message ", string(message.Contents), "\r")
    })

// Prepare message to send to private topic
submitMessageTx, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Submitkey set!")).
    SetTopicID(topicID).
    FreezeWith(client)

if err != nil {
    println(err.Error(), ": error freezing topic message submit
transaction")
    return
}

// Sign message with submit key
submitMessageTx.Sign(myPrivateKey)

```

```

// Submit message
submitTxResponse, err := submitMessageTx.Execute(client)
if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

// Get the receipt of the transaction
receipt, err := submitTxResponse.GetReceipt(client)

// Get the transaction status
transactionStatus := receipt.Status
fmt.Println("The message transaction status " +
transactionStatus.String())

// Prevent the program from exiting to display the message from the mirror
node to the console
time.Sleep(30000)
}

```

</details>

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Michiel, Developer Advocate</p><p><a
href="https://github.com/michielmulders">GitHub</a> | <a
href="https://www.linkedin.com/in/michielmulders/">LinkedIn</a></p></td><td><a
href="https://www.linkedin.com/in/michielmulders/">https://www.linkedin.com/in/
michielmulders/</a></td></tr><tr><td align="center"><p>Editor: Krystal,
Technical Writer</p><p><a href="https://github.com/theekrystallee">GitHub</a> |
<a href="https://twitter.com/theekrystallee">Twitter</a></p></td><td><a
href="https://twitter.com/theekrystallee">https://twitter.com/theekrystallee</
a></td></tr></tbody></table>

```

submit-your-first-message.md:

Submit Your First Message

Summary

With the Hedera Consensus Service (HCS), you can develop applications like stock markets, audit logs, stablecoins, or new network services that require high throughput and decentralized trust. This is made possible by having direct access to the native speed, security, and fair ordering guarantees of the Hashgraph consensus algorithm, with the full trust of the Hedera ledger.

In short, HCS offers the validity of the order of events and transparency into the history of events without requiring a persistent history of transactions. To achieve this, Mirror nodes store all transaction data so you can retrieve it to audit events.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the

previous examples.

Get a Hedera testnet account.
Set up your environment here.

✔ You can find a full code check for this tutorial at the bottom of this page.

Table of Contents

1. Create Topic
2. Subscribe to Topic
3. Submit Message
4. Code Check

1. Create your first topic

To create your first topic, you will use the `TopicCreateTransaction()`, set its properties, and submit it to the Hedera network. In this tutorial, you will create a public topic by not setting any properties on the topic. This means that anyone can send messages to your topic.

If you would like to create a private topic, you can optionally set a topic key (`setSubmitKey()`). This means that messages submitted to this topic require the topic key to sign. If the topic key does not sign a message, the message will not be submitted to the topic.

After submitting the transaction to the Hedera network, you can obtain the new topic ID by requesting the receipt. Creating a topic only costs you \$0.01.

```
{% tabs %}
{% tab title="Java" %}
java
// Create a new topic
TransactionResponse txResponse = new TopicCreateTransaction()
    .execute(client);

// Get the receipt
TransactionReceipt receipt = txResponse.getReceipt(client);

// Get the topic ID
TopicId topicId = receipt.topicId;

// Log the topic ID
System.out.println("Your topic ID is: " +topicId);

// Wait 5 seconds between consensus topic creation and subscription creation
Thread.sleep(5000);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create a new topic
let txResponse = await new TopicCreateTransaction().execute(client);

// Grab the newly generated topic ID
let receipt = await txResponse.getReceipt(client);
let topicId = receipt.topicId;
console.log(Your topic ID is: ${topicId});
```

```

// Wait 5 seconds between consensus topic creation and subscription creation
await new Promise((resolve) => setTimeout(resolve, 5000));

{% endtab %}

{% tab title="Go" %}
go
// Create a new topic
transactionResponse, err := hedera.NewTopicCreateTransaction().
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating topic")
    return
}

// Get the topic create transaction receipt
transactionReceipt, err := transactionResponse.GetReceipt(client)

if err != nil {
    println(err.Error(), ": error getting topic create receipt")
    return
}

// Get the topic ID from the transaction receipt
topicID := transactionReceipt.TopicID

//Log the topic ID to the console
fmt.Printf("topicID: %v\n", topicID)

{% endtab %}
{% endtabs %}

```

2. Subscribe to a topic

After you create the topic, you will want to subscribe to the topic via a Hedera mirror node. Subscribing to a topic via a Hedera mirror node allows you to receive the stream of messages that are being submitted to it.

```

{% hint style="info" %}
The Hedera Testnet client already establishes a connection to a Hedera mirror
node. You can set a custom mirror node by calling <mark
style="color:blue;">client.SetMirrorNetwork()</mark>. Please note that you can
subscribe to Hedera Consensus Service (HCS) topics via gRPC API only. Remember
to set the mirror node's host and port accordingly when dealing with another
mirror node provider.
{% endhint %}

```

To subscribe to a topic, you will use <mark style="color:purple;">TopicMessageQuery()</mark>. You will provide it with the topic ID to subscribe to, the Hedera mirror node client information, and the topic message contents to return.

```

{% tabs %}
{% tab title="Java" %}
java
// Subscribe to the topic
new TopicMessageQuery()
    .setTopicId(topicId)
    .subscribe(client, resp -> {
        String messageAsString = new String(resp.contents,

```

```

StandardCharsets.UTF8);
        System.out.println(resp.consensusTimestamp + " received topic
message: " + messageAsString);
    });

{% endtab %}

{% tab title="JavaScript" %}
<pre class="language-javascript"><code class="lang-javascript"><strong>//
Subscribe to the topic
</strong><strong>new TopicMessageQuery()
</strong> .setTopicId(topicId)
    .subscribe(client, null, (message) => {
        let messageAsString = Buffer.from(message.contents, "utf8").toString();
        console.log(
            `${message.consensusTimestamp.toString()} Received: ${messageAsString}
        );
    });
</code></pre>
{% endtab %}

{% tab title="Go" %}
go
// Subscribe to the topic
, err = hedera.NewTopicMessageQuery().
    SetTopicID(topicID).
    Subscribe(client, func(message hedera.TopicMessage) {
        fmt.Println(message.ConsensusTimestamp.String(), "received topic
message ", string(message.Contents), "\r")
    })

{% endtab %}
{% endtabs %}

```

3. Submit a message

Now you are ready to submit your first message to the topic. To do this, you will use `TopicMessageSubmitTransaction()`. For this transaction, you will provide the topic ID and the message to submit to it. Each message you send to a topic costs you \$0.0001. In other words, you can send 10,000 messages for \$1 on the Hedera Network.

```

{% tabs %}
{% tab title="Java" %}
java
// Send message to the topic
TransactionResponse submitMessage = new TopicMessageSubmitTransaction()
    .setTopicId(topicId)
    .setMessage("Hello, HCS!")
    .execute(client);

// Get the receipt of the transaction
TransactionReceipt receipt2 = submitMessage.getReceipt(client);

// Prevent the main thread from exiting so the topic message can be returned and
printed to the console
Thread.sleep(30000);

{% endtab %}

{% tab title="JavaScript" %}
javascript

```

```

// Send message to the topic
let sendResponse = await new TopicMessageSubmitTransaction({
    topicId: topicId,
    message: "Hello, HCS!",
}).execute(client);

// Get the receipt of the transaction
const getReceipt = await sendResponse.getReceipt(client);

// Get the status of the transaction
const transactionStatus = getReceipt.status
console.log("The message transaction status " + transactionStatus.toString())

{% endtab %}

{% tab title="Go" %}
go
// Send message to the topic
submitMessage, err := hedera.NewTopicMessageSubmitTransaction().
    SetMessage([]byte("Hello, HCS!")).
    SetTopicID(topicID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error submitting to topic")
    return
}

// Get the receipt of the transaction
receipt, err := submitMessage.GetReceipt(client)

// Get the transaction status
transactionStatus := receipt.Status
fmt.Println("The message transaction status " + transactionStatus.String())

// Prevent the program from exiting to display the message from the mirror node
to the console
time.Sleep(30000)

{% endtab %}
{% endtabs %}

To conclude: The total cost to create a topic and send a message to it is
$0.0101.

```

Code Check ☒

<details>

<summary>Java</summary>

```

java
import com.hedera.hashgraph.sdk.;
import io.github.cdimascio.dotenv.Dotenv;

import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeoutException;

public class CreateTopicTutorial {
    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException, InterruptedException {

```



```

        // Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        // Build your Hedera client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Create a new topic
        TransactionResponse txResponse = new TopicCreateTransaction()
            .execute(client);

        // Get the receipt
        TransactionReceipt receipt = txResponse.getReceipt(client);

        // Get the topic ID
        TopicId topicId = receipt.topicId;

        // Log the topic ID
        System.out.println("Your topic ID is: " +topicId);

        // Wait 5 seconds between consensus topic creation and subscription
creation
        Thread.sleep(5000);

        // Subscribe to the topic
        new TopicMessageQuery()
            .setTopicId(topicId)
            .subscribe(client, resp -> {
                String messageAsString = new String(resp.contents,
StandardCharsets.UTF8);
                System.out.println(resp.consensusTimestamp + " received
topic message: " + messageAsString);
            });

        // Send message to topic
        TransactionResponse submitMessage = new TopicMessageSubmitTransaction()
            .setTopicId(topicId)
            .setMessage("Hello, HCS!")
            .execute(client);

        // Get the receipt of the transaction
        TransactionReceipt receipt2 = submitMessage.getReceipt(client);

        // Wait before the main thread exits to return the topic message to the
console
        Thread.sleep(30000);
    }
}

```

</details>

<details>

<summary>JavaScript</summary>

```

<pre class="language-javascript"><code class="lang-javascript">console.clear();
require("dotenv").config();
const {
    AccountId,

```

```

    PrivateKey,
    Client,
    TopicCreateTransaction,
<strong>    TopicMessageQuery,
</strong>    TopicMessageSubmitTransaction,
} = require("@hashgraph/sdk");

// Grab the OPERATORID and OPERATORKEY from the .env file
const myAccountId = process.env.MYACCOUNTID;
const myPrivateKey = process.env.MYPRIVATEKEY;

// Build Hedera testnet and mirror node client
const client = Client.forTestnet();

// Set the operator account ID and operator private key
client.setOperator(myAccountId, myPrivateKey);

async function submitFirstMessage() {
    // Create a new topic
    let txResponse = await new TopicCreateTransaction().execute(client);

    // Grab the newly generated topic ID
    let receipt = await txResponse.getReceipt(client);
    let topicId = receipt.topicId;
    console.log(Your topic ID is: ${topicId});

    // Wait 5 seconds between consensus topic creation and subscription creation
    await new Promise((resolve) => setTimeout(resolve, 5000));

    // Create the topic
    new TopicMessageQuery()
        .setTopicId(topicId)
        .subscribe(client, null, (message) => {
            let messageAsString = Buffer.from(message.contents, "utf8").toString();
            console.log(
                ${message.consensusTimestamp.toString()} Received: ${messageAsString}
            );
        });

    // Send message to topic
    let sendResponse = await new TopicMessageSubmitTransaction({
        topicId: topicId,
        message: "Hello, HCS!",
    }).execute(client);
    const getReceipt = await sendResponse.getReceipt(client);

    // Get the status of the transaction
    const transactionStatus = getReceipt.status;
    console.log("The message transaction status: " +
transactionStatus.toString());
}

```

```

submitFirstMessage();
</code></pre>

```

</details>

<details>

<summary>Go</summary>

```

go
package main

```

```

import (
    "fmt"
    "os"
    "time"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {
    // Loads the .env file and throws an error if it cannot load the variables
    // from that file correctly
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%v\n", err))
    }

    // Grab your testnet account ID and private key from the .env file
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    // Create your testnet client
    client := hedera.ClientForTestnet()
    client.SetOperator(myAccountId, myPrivateKey)

    // Create a new topic
    transactionResponse, err := hedera.NewTopicCreateTransaction().
        Execute(client)

    if err != nil {
        println(err.Error(), ": error creating topic")
        return
    }

    // Get the topic create transaction receipt
    transactionReceipt, err := transactionResponse.GetReceipt(client)

    if err != nil {
        println(err.Error(), ": error getting topic create receipt")
        return
    }

    // Get the topic ID from the transaction receipt
    topicID := transactionReceipt.TopicID

    // Log the topic ID to the console
    fmt.Printf("topicID: %v\n", topicID)

    // Create the query to subscribe to a topic
    , err = hedera.NewTopicMessageQuery().
        SetTopicID(topicID).
        Subscribe(client, func(message hedera.TopicMessage) {
            fmt.Println(message.ConsensusTimestamp.String(), "received
topic message ", string(message.Contents), "\r")

```

```

    })

    // Submit message to topic
    submitMessage, err := hedera.NewTopicMessageSubmitTransaction().
        SetMessage([]byte("Hello, HCS!")).
        SetTopicID(topicID).
        Execute(client)

    if err != nil {
        println(err.Error(), ": error submitting to topic")
        return
    }

    // Get the transaction receipt
    receipt, err := submitMessage.GetReceipt(client)

    // Log the transaction status
    transactionStatus := receipt.Status
    fmt.Println("The transaction message status " +
transactionStatus.String())

    // Prevent the program from exiting to display the message from the
    mirror to the console
    time.Sleep(30 * time.Second)
}

```

</details>

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden></th><th data-hidden></th><th data-hidden
data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Simi, Sr. Software Manager </p><p><a
href="https://github.com/ed-marquez">GitHub</a> | <a
href="https://www.linkedin.com/in/shunjan">LinkedIn</a></p></td><td></td><td></
td><td><a href="https://www.linkedin.com/in/shunjan">https://www.linkedin.com/
in/shunjan </a></td></tr><tr><td align="center"><p>Editor: Michiel, Developer
Advocate</p><p><a href="https://github.com/michielmulders">GitHub</a> | <a
href="https://www.linkedin.com/in/michielmulders/">LinkedIn</a></p></td><td></
td><td></td><td><a
href="https://www.linkedin.com/in/michielmulders/">https://www.linkedin.com/in/
michielmulders/</a></td></tr></tbody></table>

```

README.md:

Getting Started with the Hedera Consensus Service Fabric Plugin

You must have a basic understanding of the Hyperledger Fabric network, its key concepts, first-network sample, and transaction flow. Please visit the Hyperledger Fabric docs to familiarize yourself with these concepts if you have not done so already.

Background

Hyperledger Fabric is one of the most popular private/permissioned enterprise blockchain frameworks available today. Fabric's modular architecture approach enables users to specify network components like network members and choice of consensus protocol \(\text{ordering service}\).

In this tutorial, you will create a Hyperledger Fabric network that leverages the Hedera Consensus Service Fabric plug-in to use Hedera Hashgraph as the ordering service via the first-network sample. The HCS Hyperledger Fabric network will be composed of two organizations. Each organization will host two peer nodes. The two organizations will privately communicate and transact within a Hyperledger Fabric channel.

If you already have experience with the Hyperledger Fabric network and the first-network sample, you can skip down to the requirements section to get started.

Transaction Flow

1. A client application creates a transaction proposal and sends it to Hyperledger Fabric network peers. The transaction proposal is endorsed \ (signed\) by the Hyperledger Fabric network peers. The endorsed transaction proposal is sent back to the client application.
2. Client application submits the endorsed transaction to the Hyperledger Fabric network HCS ordering node. That Hyperledger Fabric ordering node interacts with the HCS fabric plugin as the ordering service
3. Fabric transactions are fragmented into many messages and submitted to the Hedera network against a particular topic ID
4. Hedera mainnet nodes timestamp and order the fragmented messages
5. Hedera mirror nodes read the fragmented messages in consensus order from the Hedera mainnet
6. Hedera mirror node relays the ordered messages back to the Hyperledger Fabric ordering node
7. Hyperledger Fabric ordering nodes reassembles the fragmented messages into fabric transactions, form fabric blocks, and communicate the blocks to the Hyperledger Fabric peer nodes
8. Hyperledger Fabric peer nodes update to have the same world state

Concepts

A basic description of the concepts for the purposes of this tutorial.

Client Application

A client application communicates to a Hyperledger Fabric peer node to submit a transaction proposal and collects the endorsed \ (signed\) transaction proposal. The transaction proposal may initiate a modification of the current state of the distributed ledger.

Hedera Consensus Service Fabric Plugin

A consensus service plugin for the Hyperledger Fabric network that leverages the Hedera Consensus Service to order transactions using the unique properties of the hashgraph consensus algorithm that powers the Hedera network.

Hedera Mirror Node

A Hedera mirror node reads the ordered messages from a Hedera mainnet node and communicates that information back to the Hyperledger Fabric network. Each message has the following information:

- ConsensusTimestamp
- Message Body
- Topic Running Hash
- Topic Sequence Number

Hedera Mainnet Node

A Hedera mainnet node receives the Fabric transaction in fragmented message transactions from the HCS Hyperledger ordering node. That mainnet node gossips the transactions to the other mainnet nodes which collectively assigns the transactions a consensus timestamp and order within the corresponding topic.

fabric-hcs Repository

The project that contains all the necessary files to run the HCS Hyperledger Fabric network sample.

Hyperledger Fabric Network

The Hyperledger Fabric network is composed of peer nodes, ordering nodes, chaincode \(\smart contracts\), organizations, channels, a distributed ledger and an ordering service.

Hyperledger Fabric Network Peers

Peers are nodes in the Hyperledger Fabric network that host instances of the ledger and instances of the chaincode. Client applications interact with the peer nodes to endorse a transaction proposal.

Hyperledger Fabric Channels

Channels are a mechanism by which organizations can communicate and transact privately with one another.

Hyperledger Fabric Organizations

Organizations are participants that would like to communicate and transact privately with one another.

Hyperledger Fabric Orderer

A node that orders the transactions received from a client application \(\also known as an "ordering node"\). In this case, the ordering node interacts with the HCS fabric plugin as the ordering service.

Requirements:

```
{% hint style="info" %}
If you would like to run the sample using a virtual environment, please follow
the instructions here.
{% endhint %}
```

Hedera Consensus Service

Testnet account ID and private key
Please follow the instructions here

pluggable-hcs Repository

You will be directed to clone this repository in the outlined steps below

The HCS Fabric plugin supports Fabric ordering service 2.0 and is compatible with older versions peers.

Hyperledger Fabric Network

Download and install the following if you do not already have them on your computer.

Git
Download Git

Check to see if you have it installed from your terminal: `git --version`
cURL
Download cURL
Check to see if you have it installed from your terminal: `curl --version`
wget
Install from your terminal `\(MacOS\):brew install wget`
Check to see if you have it installed from your terminal: `wget --version`
Docker and Docker Compose
Download Docker
Docker version 17.06.2-ce or greater is required
Check to see if you have it installed from your terminal: `docker --version && docker-compose --version`
Go Programming Language
Download Go
Go version 1.13.x is required
Check to see if you have it installed from your terminal: `go version`
make
Install from your terminal `\(MacOS\): brew install make`
Check to see if you have installed from your terminal: `make --version`
gcc
Check to see if you have it installed from your terminal: `gcc --version`

Additionally, you may reference Hyperledger Fabric's documentation for the prerequisites.

Terminal/IDE

You should be able to use the commands provided in this tutorial in terminal prompt or IDE of choice.

1. Clone the pluggable-hcs Project

Open your terminal or favorite IDE
Enter the following commands to set-up your environment variables `\(required\)`
Optionally you can make these settings permanent by placing them in the appropriate startup file, such as your personal `/.bashrc` file if you are using the bash shell under Linux.

```
text
$ export GOPATH=$HOME/go
$ export PATH=$PATH:$GOPATH/bin
```

Create a hyperledger directory and navigate into that directory

```
text
$ mkdir -p $GOPATH/src/github.com/hyperledger && cd
$GOPATH/src/github.com/hyperledger
```

Clone the pluggable-hcs repository and rename it to fabric
You must rename the folder to fabric otherwise you will run into issues in the following steps

```
text
$ git clone https://github.com/hyperledger-labs/pluggable-hcs fabric
$ cd fabric
```

Confirm you are on the master branch

```
text
$ git branch
```

☆ You have now successfully set your Go path variables and installed the pluggable-hcs/fabric repository.

2. Build Fabric Binaries and Docker Images

cd to the fabric directory if you are not there already from your terminal or favorite IDE

```
text
$ cd fabric
```

Follow the commands below to build the required fabric binaries and docker images

Note: This process may take a few minutes to complete

```
text
$ make clean
$ make configtxgen configtxlator cryptogen orderer peer docker
```

3. Hedera Network & HCS Hyperledger Fabric Orderer Configuration

You will now enter your Hedera testnet account ID and private key information to the relevant configuration files. If you have not previously generated your testnet account, please follow the instructions here.

Navigate to the first-network directory from your terminal or favorite IDE

```
text
$ cd first-network
```

Edit the hedera\env.json file via the terminal or IDE

This sets up the Hedera environment and allows you to create topics and submit messages to the Hedera network. The Hedera account entered here pays for the transaction fees associated with creating and submitting messages.

If you are using an IDE, you may skip this command and edit the file

```
text
$ nano hederaenv.json
```

Enter your Hedera account ID \ (e.g. 0.0.1234\) in the operatorId field
Enter your account private key in the operatorKey field

```
javascript
{
  "operatorId": "put your testnet account id here",
  "operatorKey": "put your account private key here",
  "nodeId": "",
  "nodeAddress": "",
  "network": {
    "0.testnet.hedera.com:50211": "0.0.3",
    "1.testnet.hedera.com:50211": "0.0.4",
    "2.testnet.hedera.com:50211": "0.0.5",
    "3.testnet.hedera.com:50211": "0.0.6"
  },
  "mirrorNodeAddress": "hcs.testnet.mirrornode.hedera.com:5600"
}
```

Save the hedera\env.json file

Open the orderer.yaml file from your terminal or favorite IDE
Here you will set the configurations for the HCS Hyperledger Fabric orderer node
This file configures hcs as the ordering service
This also contains the node address and node ID to interact with the Hedera mainnet nodes
The Hedera mirror node address to receive ordered transaction from
If you are using an IDE, you may skip this command and edit the file

```
text
$ nano orderer.yaml
```

Scroll down to the "SECTION: Hcs" heading
Enter your Hedera account ID in the Operator.Id field
Enter your Hedera account private key in the Operator.PrivateKey.Key field

```
yaml
operator set for the orderer
Operator:
  Id: Your Hedera testnet account id here
  PrivateKey:
    type of the private key
    Type: ed25519
    key string (hex or PEM encoded) or path to the key file
    Key: Your Hedera testnet account private key string here
```

Save the orderer.yaml file

Please make sure you have entered your Hedera information correctly with no syntax errors as it will cause issues when trying to run the network later.

☆ You have now successfully set up your configuration variables for the Hedera operator and for the HCS Hyperledger ordering node.

4. Run Your Network

In this step you will create your HCS Hyperledger Fabric Network.

Make sure you are within the first-network directory before running these commands.

Enter the following command to start the HCS Hyperledger Fabric network

```
text
$ ./byfn.sh up -t 20
```

Enter "Y" to accept the 20 second timeout and CLI delay of 3 seconds

```
{% hint style="info" %}
Please note it's necessary to set the timeout to 20 seconds since otherwise some
checks in the script may fail due to HCS having a larger consensus delay than
raft/kafka based ordering service.
{% endhint %}
```

The script generates two HCS topics
One topic will be for the Hyperledger Fabric system channel
One topic will be for the Hyperledger Fabric application channel
HCS topics in this example are configured so that anyone can submit messages to them
HCS messages can be configured to be private topics where the sender would

require the submitKey to successfully publish messages to them

Transactions submitted to either of these channels will be visible from a mirror node explorer

Transactions can be fragmented into smaller chunks resulting in multiple HCS messages for a single transaction

The sample is almost identical with the upstream first-network sample as of 2.0.0 release. In summary, the modifications are:

1. Updated docker compose files to use hcs-based orderers.
2. Added a function in byfn.sh to generate a 256-bit AES key for data encryption/decryption between fabric orderers and hedera testnet.
3. Added a function in byfn.sh to use the hcscli tool for HCS topic creation and mapping topics to fabric channels.

A successful run will end with the following message:

```
===== All GOOD, BYFN execution completed =====
```

```
| | | \ | | | \
| | | | \ | | | |
| | | | \ | | | |
|| || \ | | /
```

5. Tear down the network

It is required to run the following command to tear down the network

If you do not tear down the network and try to restart the network you may run into issues

text

```
$ ./byfn.sh down
```

6. Verify Topics & Messages

Topics and messages created in this tutorial can be verified on any available mirror node explorer

At the start of the script, you can see the two HCS topic IDs that were created

installing hcscli ...

generated HCS topics: 0.0.23419 0.0.23420

0.0.23419 will be used for the system channel, and 0.0.23420 will be used for the application channel

Visit a Hedera mirror node explorer to verify the topics and messages that were created on testnet by searching the two topic IDs

You will be able to view the application channel and system channel topics and all associated messages from this example

A single Fabric transaction sent to an ordering node could result in multiple HCS consensus messages as HCS messages have a 6k message size limit

i.e. there may not be a 1:1 correlation between a Fabric transaction and HCS message

A fabric transaction payload is encrypted by the ordering node therefore the subsequent HCS transaction payload is also encrypted

All messages on the mirror node explorer will be displayed in encrypted format.

Make sure you have selected the testnet network toggle in the explorer as the topics and messages created through this tutorial will not appear on the main network.

You have successfully done the following:

Created a Hyperledger Fabric network using the HCS Fabric plugin as the ordering service

Verified the topics and messages created in this example network

Running into issues or have suggestions? Visit the developer advocates in Discord and post your comments to the hedera-consensus-service channel 🤖 .

Have a question?

Ask it on StackOverflow

virtual-environment-set-up.md:

Virtual Environment Set-up

Enables developers to run the HCS Hyperledger Fabric sample network using a virtual environment set-up.

Requirements

Hedera testnet account ID and account private key

pluggable-hcs repository

Vagrant

Virtual Box

Terminal/IDE

1. Open your terminal/IDE and CD to where you would like to clone the fabric-hcs project

Clone the pluggable-hcs repository and rename the project folder to fabric

Navigate to the fabric folder

text

```
git clone https://github.com/hyperledger-labs/pluggable-hcs fabric
```

```
cd fabric
```

You should now be in the fabric project folder

2. Confirm you are on the master branch

text

```
git branch
```

3. Navigate to the vagrant folder and start your virtual machine

text

```
cd vagrant
```

```
vagrant up
```

```
vagrant ssh
```

You should now be back in the fabric folder

Now you have your virtual environment ready to go. Please refer to step two: Build Fabric Binaries and Docker Images in the master tutorial to continue.

```
{% page-ref page="./" %}
```

```
{% hint style="info" %}
```

Have a question?

Ask it on StackOverflow

```
{% endhint %}
```

create-fund-account.md:

description: >-

Hello World sequence: Create a new account on Hedera Testnet, and fund it. Do this before any of the other Hello World sequences.

Create and fund account

Why you need to create and fund an account

Hedera is a distributed ledger technology (DLT). To interact with it, you will need to send transactions to the network, which will then process them and add them to the ledger if they are deemed to be valid. On most web services (web2), you need to authenticate using usernames and passwords to operate your account. On DLTs such as Hedera, it is similar, except that you will need to use cryptographic keys instead of passwords to operate your account. One key difference is that unlike web2, each interaction needs to be paid for using the native currency of the DLT, which is similar to micro-transactions. On Hedera, this currency is HBAR.

What you will accomplish

- [] Generate cryptographic keys to be used by a Hedera account
- [] Use the Hedera Faucet to create and fund a new account with Testnet HBAR

Prerequisites

Before you begin, you should be familiar with the following:

- [x] JavaScript syntax

<details>

<summary>Also, you should have the following set up on your computer</summary>

- [x] POSIX-compliant shell

For Linux & Mac: The shell that ships with the operating system will work. Either bash or zsh will work.

For Windows: The shells that ship with the operating system (cmd.exe, powershell.exe) will not work.

Recommended: git-bash which ships with git-for-windows. Install Git for Windows (Git for Windows)

Recommended (alternative): Windows Subsystem for Linux. Install WSL (Microsoft)

- [x] git installed

Minimum version: 2.37

Recommended: Install Git (Github)

- [x] A code editor or IDE

Recommended: VS Code. Install VS Code (Visual Studio)

- [x] NodeJs + npm installed

Minimum version of NodeJs: 18

Minimum version of npm: 9.5

Recommended for Linux & Mac: nvm

Recommended for Windows: nvm-windows

</details>

Get started

Set up project

To follow along, start with the main branch, which is the default branch of this repo. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

```
shell
git clone https://github.com/hedera-dev/hello-future-world.git
```

<details>

<summary>Alternative with git and SSH</summary>

If you have configured SSH to work with git, you may wish use this command instead:

```
shell
git clone git@github.com:hedera-dev/hello-future-world.git
```

</details>

In the terminal, from the hello-future-world directory, enter the subdirectory for this sequence.

```
shell
cd 00-create-fund-account/
```

Install the dependencies using npm.

```
shell
npm install
```

Create your .env file

Make a .env file by copying the provided .env.sample file. Then open the .env file in a code editor, such as VS Code.

```
shell
cp .env.sample .env
```

Checkpoint

<details>

<summary>Check your prerequisites and project set up</summary>
❗</summary>

In your terminal, enter the following command.

```
sh
node checkpoint-setup.js
```

This script checks multiple set up related items, and should produce output like

this:

```
git check:
OK!
git version check:
OK!
node version check:
OK!
npm version check:
OK!
npm install check:
OK!
.env file check:
OK!
shell check:
OK!
```

If the output contains any errors, please address them before continuing with the rest of this sequence.

</details>

Generate seed phrase

In the terminal, run the following command:

```
shell
npx mnemonics@1.1.3
```

This should output a seed phrase, a list of 12 randomly selected dictionary words, for example:

artefact gasp crop double silk grid visual gather argue glow melody net

<details>

<summary>Alternative way to generate a seed phrase</summary>

```
Visit https://iancoleman.io/bip39/
Select 12 from the dropdown next to the GENERATE button
Press the GENERATE button
Copy the seed phrase from the text field labelled BIP39 Mnemonic.
```

</details>

{% hint style="info" %}

Note that mnemonics is a tool that generates BIP-39 seed phrases, and your seed phrase will be different from above.

If this is your first time running this command, you need to enter y to agree to do so:

```
Need to install the following packages:
mnemonics@1.1.3
Ok to proceed? (y)
```

This seed phrase will be used to generate the cryptographic keys for the

accounts that you are about to create.
{% endhint %}

Copy the seed phrase. Replace SEEDPHRASE in the .env file with it. The file contents should now look similar to this:

```
{% code title=".env" overflow="wrap" %}  
shell  
SEEDPHRASE="artefact gasp crop double silk grid visual gather argue glow melody  
net"  
ACCOUNTPRIVATEKEY=YOURHEXENCODEDPRIVATEKEY  
ACCOUNTID=YOURACCOUNTID  
RPCURL=YOURJSONRPCURL  
  
{% endcode %}
```

You do not need to modify the other values in the .env file yet.

Be sure to save your files before moving on to the next step.

Write the script

An almost-complete script has already been prepared for you, and you will only need to make a few modifications (outlined below) for it to run successfully.

Open the script file, script-create-fund-account.js, in a code editor.

```
{% hint style="info" %}  
To follow along in the tutorial, when asked to modify code, look for a comment  
to locate the specific lines of code which you will need to edit.
```

For example, the comment for Step 1 looks like this:

```
javascript  
    // Step (1) in the accompanying tutorial  
  
{% endhint %}
```

Step 1: Derive private key

The ethersHdNode module has been imported from EthersJs. This takes a seed phrase as input, and outputs a private key. To do so, invoke the fromMnemonic() method and pass in process.env.SEEDPHRASE as the parameter:

```
{% code title="script-create-fund-account.js" overflow="wrap" %}  
javascript  
    const hdNodeRoot = ethersHdNode.fromMnemonic(process.env.SEEDPHRASE);  
  
{% endcode %}  
  
{% hint style="info" %}  
You will need to delete the inline comment that looks like this: / ... /.  
Replace it with the correct code. For example, in this step, the change looks  
like this:
```

```
diff  
-    const hdNodeRoot = ethersHdNode.fromMnemonic(/ ... /);  
+    const hdNodeRoot = ethersHdNode.fromMnemonic(process.env.SEEDPHRASE);  
  
{% endhint %}
```

The hdNodeRoot instance is subsequently used to generate the private keys.

Step 2: Derive EVM address

A `privateKey` instance has been initialized. This requires no further input to derive an EVM address - read its `publicKey` property, and then invoke its `toEvmAddress` method:

```
{% code title="script-create-fund-account.js" overflow="wrap" %}
javascript
  const evmAddress = 0x${privateKey.publicKey.toEvmAddress()};

{% endcode %}
```

Run the script

In the terminal, run the script using the following command:

```
shell
node script-create-fund-account.js
```

This should produce output similar to the following:

```
privateKeyHex:
0x0ac20a3c1573ba9a5c6c69349fa51f40bd502cf250e226a7100869338f15aae2
evmAddress: 0x61b47b6aa6595a6546873fc831331f36639c906f
accountExplorerUrl:
https://hashscan.io/testnet/account/0x61b47b6aa6595a6546873fc831331f36639c906f
accountId: undefined
accountBalanceHbar: undefined
```

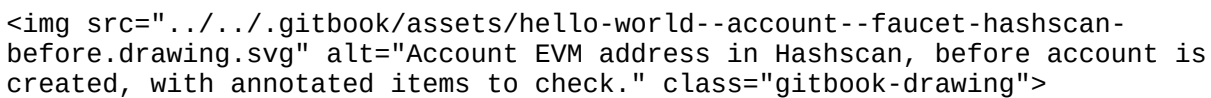
Note that `accountId` and `accountBalanceHbar` are both undefined, because generating cryptographic keys alone is not enough to create an account - that only happens upon the first transaction.

Copy the value of `privateKeyHex`. Replace `YOURHEXENCODEDPRIVATEKEY` in the `.env` file with it. The file contents should now look similar to this:

```
{% code title=".env" overflow="wrap" %}
shell
SEEDPHRASE="artefact gasp crop double silk grid visual gather argue glow melody
net"
ACCOUNTPRIVATEKEY=0x0ac20a3c1573ba9a5c6c69349fa51f40bd502cf250e226a7100869338f15
aae2
ACCOUNTID=YOURACCOUNTID
RPCURL=YOURJSONRPCURL

{% endcode %}
```

Copy the value of `accountExplorerUrl` and visit this in your browser.

The image shows a screenshot of a web browser displaying the Hashscan interface. It features a drawing of a faucet and an account ID, with annotations indicating where to check for the account's creation and EVM address.

You should see a page with:

- The title "Inactive EVM Address" (1)
- "Account ID: Assigned upon activation" (2)
- "EVM Address:" matching the value of `evmAddress` output earlier (3)

A helpful hint saying "Own this account? Activate it by transferring any amount of h or tokens to ..." (4)

This is precisely the next step!

Transfer and activate account

Visit portal.hedera.com/faucet.

```
{% hint style="info" %}
```

The faucet dispenses Testnet HBAR to any account on Hedera Testnet. When it is asked to dispense to an EVM address that does not yet have an account, the account gets created as part of the HBAR transfer transaction.

```
{% endhint %}
```

```

```

Paste the value of `evmAddress` output earlier into the "enter wallet address" field (1)

Press the "receive testnet HBAR" button (2)

A confirmation dialog will pop up.

```

```

Complete the ReCaptcha (1)

Press the "confirm transaction" button. (2)

A success dialog will pop up.

```

```

The account ID is displayed (1)

This indicates that the Testnet HBAR has been transferred, and in the process a new account has been created.

Note that the EVM address is not the same as the account ID - instead the EVM address is an alias of the account ID.

Press the icon to copy the account ID (2)

Replace `YOURACCOUNTID` in the `.env` file with it. The file contents should now look similar to this:

```
{% code title=".env" overflow="wrap" %}
```

```
shell
```

```
SEEDPHRASE="artefact gasp crop double silk grid visual gather argue glow melody net"
```

```
ACCOUNTPRIVATEKEY=0x0ac20a3c1573ba9a5c6c69349fa51f40bd502cf250e226a7100869338f15 aae2
```

```
ACCOUNTID=0.0.2667268
```

```
RPCURL=YOURJSONRPCURL
```

```
{% endcode %}
```

```
{% hint style="info" %}
```

You may have noticed that `RPCURL` is unused throughout. This is intentional - as it will be used in some of the other Hello World sequences, so be sure to check

them out after completing this one!
{% endhint %}

Refresh the Hashscan page in your browser. Note that this is the accountExplorerUrl that was output from the previous run of the script. This time you should see:

```

```

The title is "Account" (1)
instead of "Inactive EVM Address"
The "Account ID" field should matching the value of ACCOUNTID above (2)
instead of "Assigned upon activation"
The "Create Transaction" field displays a transaction ID (3)

Verify account creation and funding

In the terminal, re-run the script using the following command:

```
shell  
node script-create-fund-account.js
```

This should produce output similar to the following:

```
privateKeyHex:  
0x0ac20a3c1573ba9a5c6c69349fa51f40bd502cf250e226a7100869338f15aae2  
evmAddress: 0x61b47b6aa6595a6546873fc831331f36639c906f  
accountExplorerUrl:  
https://hashscan.io/testnet/account/0x61b47b6aa6595a6546873fc831331f36639c906f  
accountId: 0.0.2667268  
accountBalanceHbar: 100.00000000
```

Note that this is almost the same as when you first ran the same script. The difference is that previously both accountId and accountBalanceHbar were undefined; and now accountId should now show a value (in the format of 0.0.XYZ), and accountBalanceHbar should now show a number (with 8 decimal places). This is because the account has been created and funded.

🎉 Now you are ready to start using your Hedera Testnet account from the portal within script files on your computer! 🎉

Complete

Congratulations, you have completed the create and fund account Hello World sequence! 🎉🎉🎉

You have learned how to:

- [x] Generate cryptographic keys to be used by a Hedera account
- [x] Use the Hedera Faucet to create and fund a new account with Testnet HBAR

Next Steps

Now that you have an account on Hedera Testnet and it is funded, you can interact with the Hedera network. Continue by following along with the other Hello World sequences.

Cheat sheet

<details>

<summary>Skip to final state</summary>

The repo, github.com/hedera-dev/hello-future-world, is intended to be used alongside this tutorial.

To skip ahead to the final state, use the completed branch. You may use this to compare your implementation to the completed steps of the tutorial.

```
shell
git fetch origin completed:completed
git checkout completed
```

Alternatively, you may view the completed branch on Github: github.com/hedera-dev/hello-future-world/tree/completed/00-create-fund-account

</details>

Writer: Brendan Editors: Abi, Michiel, Ryan, Krystal

hcs-topic.md:

```
---
description: >-
  Hello World sequence: Create a new topic Hedera Consensus Service (HCS), and
  publish a message to this topic.
---
```

HFS: Files

What you will accomplish

- [] Create a new Topic on HCS
- [] Publish a new message to this Topic

Prerequisites

Before you begin, you should have completed the following Hello World sequence:

- [X] create-fund-account.md

Get started: Set up project

To follow along, start with the main branch, which is the default branch of the repo. This gives you the initial state from which you can follow along with the

steps as described in the tutorial.

{% hint style="warning" %}

You should already have this from the "Create and Fund Account" sequence. If you have not completed this, you are strongly encouraged to do so.

Alternatively, you may wish to create a .env file and populate it as required.

{% endhint %}

In the terminal, from the hello-future-world directory, enter the subdirectory for this sequence.

```
shell
cd 06-hcs-topic/
```

Reuse the .env file by copying the one that you have previously created into the directory for this sequence.

```
shell
cp ../00-create-fund-account/.env ./
```

<details>

<summary>Check that you have copied the .env file correctly</summary>

To do so, use the pwd command to check that you are indeed in the right subdirectory within the repo.

```
shell
pwd
```

This should output a path that ends with /hello-future-world/06-hcs-topic. If not, you will need to start over.

```
/some/path/hello-future-world/06-hcs-topic
```

Next, use the ls command to check that the .env file has been copied into this subdirectory.

```
shell
ls -a
```

The first few line of the output should look display .env. If not, you'll need to start over.

```
.
..
.env
```

</details>

Next, install the dependencies using npm.

```
shell
npm install
```

Then open the script-hcs-topic.js file in a code editor, such as VS Code.

Write the script

An almost complete script has already been prepared for you, script-hcs-topic.js, and you will only need to make a few modifications (outlined below) to run successfully.

Step 1: Create a topic

To create a new HCS topic, submit a TopicCreateTransaction, from the Hedera SDK. This transaction does not need any input parameters.

```
{% code title="script-hcs-topic.js" overflow="wrap" %}
js
    const topicCreateTx = await
        new TopicCreateTransaction()
        .freezeWith(client);
```

```
{% endcode %}
```

Note that once this transaction has been completed, it is important to extract the topicID from it, as you will need this to submit messages to it. This has already been done, and no modification is necessary.

```
{% code title="script-hcs-topic.js" overflow="wrap" %}
js
    const topicId = topicCreateTxReceipt.topicId;
```

```
{% endcode %}
```

Step 2: Publish message to topic

Once the topic has been registered, you're ready to submit messages to it. Messages may be any data up to 1024 bytes.

To do so, submit a TopicMessageSubmitTransaction from the Hedera SDK. This transaction needs 2 input parameters, topicId and message. Use the topicId obtained from the receipt of the TopicCreateTransaction submitted earlier. For message, use a string Hello HCS - followed by your name (or nickname). For example, if you used bguiz, the string should be Hello HCS - bguiz.

```
{% code title="script-hcs-topic.js" overflow="wrap" %}
js
    const topicMsgSubmitTx = await
        new TopicMessageSubmitTransaction({
            topicId: topicId,
            message: 'Hello HCS - bguiz',
        })
        .freezeWith(client);
```

```
{% endcode %}
```

Run the script

In the terminal, run the script using the following command:

```
shell
```

```
node script-hcs-topic.js
```

You should see output similar to the following:

```
text
Topic created. Waiting a few seconds for propagation...
accountId: 0.0.3643569
topicId: 0.0.3791978
topicExplorerUrl: https://hashscan.io/testnet/topic/0.0.3791978
topicCreateTxId: 0.0.3643569@1711530800.714890962
topicMsgSubmitTxId: 0.0.3643569@1711530810.343147411
topicMessageMirrorUrl:
https://testnet.mirrornode.hedera.com/api/v1/topics/0.0.3791978/messages/1
```

To verify that both the TopicCreateTransaction and TopicMessageSubmitTransaction have worked, check Hashscan (the network explorer). To do so, open `topicExplorerUrl` in your browser and check that:

```

```

The topic exists, and its topic ID matches `topicId` output by the script. (1)
There is one entry in the topic, and its message is Hello HCS - followed by your name/ nickname. (2)

Complete

Congratulations, you have completed the Hedera Consensus Service Hello World sequence! 🎉🎉🎉

You have learned how to:

- [x] Create a new Topic on HCS
- [x] Publish a new message to this Topic

Next Steps

Now that you have completed this Hello World sequence, you have interacted with Hedera Consensus Service (HCS). There are other Hello World sequences for Hedera Smart Contract Service (HSCS), Hedera File Service (HFS), and Hedera Token Service (HTS), which you may wish to check out next.

Cheat sheet

```
<details>
```

```
<summary>Skip to final state</summary>
```

The repo, github.com/hedera-dev/hello-future-world, is intended to be used alongside this tutorial.

To skip ahead to the final state, use the completed branch. You may use this to compare your implementation to the completed steps of the tutorial.

```
shell
git fetch origin completed:completed
```

git checkout completed

Alternatively, you may view the completed branch on Github: github.com/hedera-dev/hello-future-world/tree/completed/06-hcs-topic

</details>

Writer: Brendan

hscs-smart-contract.md:

description: >-

Hello World sequence: Write a smart contract in Solidity, compile it, then use Hedera Smart Contract Service (HSCS) to deploy it and interact with it.

HSCS: Smart Contract

What you will accomplish

- [] Write a smart contract
- [] Compile the smart contract
- [] Deploy a smart contract
- [] Update smart contract state
- [] Query smart contract state

Prerequisites

Before you begin, you should have completed the following Hello World sequence:

create-fund-account.md

Get started: Set up project

To follow along, start with the main branch, which is the default branch of the repo. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

{% hint style="warning" %}

You should already have this from the "Create and Fund Account" sequence. If you have not completed this, you are strongly encouraged to do so.

Alternatively, you may wish to create a .env file and populate it as required.

{% endhint %}

In the terminal, from the hello-future-world directory, enter the subdirectory for this sequence.

shell

cd 03-hscs-smart-contract-ethersjs/

Reuse the .env file by copying the one that you have previously created into the directory for this sequence.

```
shell
cp ../00-create-fund-account/.env ./
```

<details>

<summary>Check that you have copied the .env file correctly</summary>

To do so, use the pwd command to check that you are indeed in the right subdirectory within the repo.

```
shell
pwd
```

This should output a path that ends with /hello-future-world/03-hscs-smart-contract-ethersjs. If not, you will need to start over.

```
/some/path/hello-future-world/03-hscs-smart-contract-ethersjs
```

Next, use the ls command to check that the .env file has been copied into this subdirectory.

```
shell
ls -a
```

The first few line of the output should look display .env. If not, you'll need to start over.

```
.
..
.env
```

</details>

Next, install the dependencies using npm. You will also need to install a Solidity compiler, using the --global flag.

```
shell
npm install && npm install --global solc@0.8.17
```

```
{% hint style="info" %}
Solidity is a programming language that was designed specifically for writing
smart contracts in. The Solidity compiler outputs bytecode that can be run by an
Ethereum Virtual Machine (EVM) implementation; including Hyperledger Besu's EVM,
which powers Hedera Smart Contract Service.
{% endhint %}
```

```
{% hint style="info" %}
Note that although the npm package is named solc, the executable exposed on your
command line is named solcjs.
{% endhint %}
```

Then, open both of the following files in a code editor, such as VS Code.

```
mycontract.sol
script-hscs-smart-contract-ethersjs.js
```


Write the smart contract

An almost-complete smart contract has already been prepared for you, `mycontract.sol`. You will only need to make one modification (outlined below) for it to compile successfully.

Step 1: Get the name stored in mapping

Within the `greet()` function, we would like to access the names mapping and retrieve the name of the account that is invoking this function. The account is identified by its EVM account alias, which is available as `msg.sender` within the Solidity code.

```
{% code title="mycontract.sol" overflow="wrap" %}
solidity
    string memory name = names[msg.sender];

{% endcode %}
```

```
{% hint style="info" %}
```

Note: This smart contract has two functions, `introduce` and `greet`. You will invoke both of them later on.

```
{% endhint %}
```

Compile the smart contract

Once you have completed writing the smart contract in Solidity, you will need to compile it using the Solidity compiler installed earlier.

Invoke the compiler on your Solidity file. Then list files in the current directory.

```
shell
solcjs --bin --abi ./mycontract.sol
ls
```

You should see an output similar to the following:

```
mycontract.sol
mycontractsolMyContract.abi
mycontractsolMyContract.bin
```

```
{% hint style="info" %}
```

The `.abi` file contains JSON and describes the interface used to interact with the smart contract.

The `.bin` file contains EVM bytecode, and this is used in the deployment of the smart contract.

Note that while the `.abi` file is human-readable, the `.bin` file is not intended to be human-readable.

```
{% endhint %}
```

Configure RPC Connection

1. Sign up for an account at auth.arkhia.io/signup. If prompted, click on the

link in your confirmation email.

2. Click on the <mark style="background-color:yellow;">+</mark><mark style="background-color:yellow;">CREATE PROJECT</mark> button in the top-right corner of the Arkhia dashboard.

3. Fill in whatever you like in the modal dialog that pops up.

4. Click on the "Manage" button on the right side of your newly created project.

Now, you should see the project details.

Under "Network", select "Hedera Testnet".

Copy the "JSON-RPC Relay" field.

In the "Security" section, copy the "API Key" field.

In the .env file, edit the property with the key RPCURL, to replace YOURJSONRPCURL with the "JSON-RPC Relay" value, followed by a / character, and finally, the "API key" value that you have just copied.

For example, if the JSON-RPC field is https://pool.arkhia.io/hedera/testnet/json-rpc/v1, and if the API key field is ABC123, the line in your .env file should look like this:

```
{% code title=".env" overflow="wrap" %}
shell
RPCURL=https://pool.arkhia.io/hedera/testnet/json-rpc/v1/ABC123

{% endcode %}

{% hint style="warning" %}
Note: Ensure that you have a / just before the API key.
{% endhint %}
```

<details>

<summary>Alternative RPC configuration</summary>

Arkhia is one of multiple options for JSON-RPC connections. This tutorial covers the available options: How to Connect to Hedera Networks Over RPC.

</details>

Write the script

An almost complete script has already been prepared for you, script-hscs-smart-contract-ethersjs.js. You will only need to make a few modifications (outlined below) for it to run successfully.

Step 2: Prepare smart contract for deployment

Initialize an instance of ContractFactory from EthersJs.

```
{% code title="script-hscs-smart-contract-ethersjs.js" overflow="wrap" %}
js
const myContractFactory = new ContractFactory(
```

```
abi, evmBytecode, accountWallet);
```

```
{% endcode %}
```

```
{% hint style="info" %}
```

Note: The ContractFactory class is used to prepare a smart contract for deployment. To do so, pass in the ABI and bytecode output by the Solidity compiler earlier. Also, pass in the accountWallet object, which is used to authorize transactions and is needed for the deployment transaction.

Upon preparation, it sends a deployment transaction to the network, and an instance of a Contract object is created based on the result of the deployment transaction. This is stored in a variable, myContract, which will be used in the next steps. This has already been done for you in the script.

```
{% endhint %}
```

Step 3: Invoke a smart contract transaction

The introduce function requires a single parameter of type string, and changes the state of the smart contract to store this value. Enter your name (or nickname) as the parameter. For example, if you wish to use "bguiz", the invocation should look like this:

```
{% code title="script-hscs-smart-contract-ethersjs.js" overflow="wrap" %}
```

```
js
```

```
    const myContractWriteTxRequest = await  
myContract.functions.introduce('bguiz');
```

```
{% endcode %}
```

Step 4: Invoke a smart contract query

In the previous step, you changed some state of the smart contract, which involved submitting a transaction to the network. This time, you are going to read some state of the smart contract. This is much simpler to do as no transaction is needed.

Invoke the greet function and save its response to a variable, myContractQueryResult. This function does not take any parameters.

```
{% code title="script-hscs-smart-contract-ethersjs.js" overflow="wrap" %}
```

```
js
```

```
    const [myContractQueryResult] = await myContract.functions.greet();
```

```
{% endcode %}
```

```
{% hint style="info" %}
```

When invoking functions in a smart contract, you may do so in two different ways:

With a transaction → Smart contract state may be changed.

Without a transaction → Smart contract state may be queried but may not be changed.

```
{% endhint %}
```

Run the script

In the terminal, run the script using the following command:

```
shell
```

```
node script-hscs-smart-contract-ethersjs.js
```

You should see output similar to the following:

```
accountId: 0.0.1201
accountAddress: 0x7394111093687e9710b7a7aEBa3BA0f417C54474
accountExplorerUrl:
https://hashscan.io/testnet/address/0x7394111093687e9710b7a7aEBa3BA0f417C54474
myContractAddress: 0x9A6856897a72E790Ae765bFF997396199BDf1B72
myContractExplorerUrl:
https://hashscan.io/testnet/address/0x9A6856897a72E790Ae765bFF997396199BDf1B72
myContractWriteTxHash:
0x32684e8d8e60126e171db968a4b20153ba96a40920a036dbebb48e19fb74664d
myContractWriteTxExplorerUrl:
https://hashscan.io/testnet/transaction/0x32684e8d8e60126e171db968a4b20153ba96a4
0920a036dbebb48e19fb74664d
myContractQueryResult: Hello future - bguiz
```

Open myContractExplorerUrl in your browser and check that:

```

```

The contract exists

Under the "Contract Bytecode" section, its "Compiler Version" field matches the version of the Solidity compiler that you used (0.8.17) (2)


Under the "Recent Contract Calls" section, There should be two transactions:

The transaction with the earlier timestamp (bottom) should be the deployment transaction. (3A)

Navigate to this transaction by clicking on the timestamp.

Under the "Contract Result" section, the "Input - Function & Args" field should be a relatively long set of hexadecimal values.

This is the EVM bytecode output by the Solidity compiler.

Navigate back to the Contract page (browser  button).

The transaction with the later timestamp (top) should be the function invocation transaction, of the introduce function. (3B)


Navigate to this transaction by clicking on the timestamp.

Under the "Contract Result" section, the "Input - Function & Args" field should be a relatively short set of hexadecimal values.

This is the representation of

the function identifier as the first eight characters (e.g. 0xc63193f6 for the introduce function), and

the input string value (e.g. 0x5626775697a0 for bguiz).

Navigate back to the Contract page (browser  button).

```
<details>
```

```
<summary>Additional checks</summary>
```

The steps above are sufficient to check that you have deployed and interacted with the same contract successfully. You may optionally wish to perform these additional checks as well.

Open myContractWriteTxExplorerUrl in your browser. This should be the same page as "the transaction with the later timestamp" in 3B from the previous checks. Check that:

The transaction exists

Its "Type" field is "ETHEREUM TRANSACTION"

Under the "Contract Result" section, its "From" field matches the value of accountId

Under the "Contract Result" section, its "To" field matches the value of

myContractAddress

</details>

Complete

Congratulations, you have completed the Hedera Smart Contract Service Hello World sequence! 🎉🎉🎉

You have learned how to:

- [x] Write a smart contract
- [x] Compile the smart contract
- [x] Deploy a smart contract
- [x] Update smart contract state
- [x] Query smart contract state

Next Steps

Now that you have completed this Hello World sequence, you have interacted with Hedera Smart Contract Service (HSCS). There are other Hello World sequences for Hedera File Service (HFS), and Hedera Token Service (HTS), which you may wish to check out next.

You may also wish to check out the more detailed HSCS workshop, which goes into greater depth.

Cheat sheet

<details>

<summary>Skip to final state</summary>

The repo, github.com/hedera-dev/hello-future-world, is intended to be used alongside this tutorial.

To skip ahead to the final state, use the completed branch. You may use this to compare your implementation to the completed steps of the tutorial.

```
shell
git fetch origin completed:completed
git checkout completed
```

Alternatively, you may view the completed branch on Github: github.com/hedera-dev/hello-future-world/tree/completed/03-hscs-smart-contract-ethersjs

</details>

Writer: Brendan Editors: Abi, Michiel, Ryan, Krystal

hts-fungible-token.md:

description: >-

Hello World sequence: Create a new fungible token using Hedera Token Service (HTS).

HTS: Fungible Token

What you will accomplish

- [] Create and mint a new fungible token on HTS
- [] Query the token balance

Prerequisites

Before you begin, you should have completed the following Hello World sequence:

create-fund-account.md

Get started: Set up project

To follow along, start with the main branch, which is the default branch of the repo. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

{% hint style="warning" %}

You should already have this from the "Create and Fund Account" sequence. If you have not completed this, you are strongly encouraged to do so.

Alternatively, you may wish to create a .env file and populate it as required.

{% endhint %}

In the terminal, from the hello-future-world directory, enter the subdirectory for this sequence.

```
shell
cd 04-hts-ft-sdk/
```

Reuse the .env file by copying the one that you have previously created into the directory for this sequence.

```
shell
cp ../00-create-fund-account/.env ./
```

<details>

<summary>Check that you have copied the .env file correctly</summary>

To do so, use the pwd command to check that you are indeed in the right subdirectory within the repo.

```
shell
pwd
```

This should output a path that ends with /hello-future-world/04-hts-ft-sdk. If not, you will need to start over.

/some/path/hello-future-world/04-hts-ft-sdk

Next, use the `ls` command to check that the `.env` file has been copied into this subdirectory.

```
shell
ls -a
```

The first few line of the output should look display `.env`. If not, you'll need to start over.

```
.
..
.env
```

</details>

Next, install the dependencies using `npm`.

```
shell
npm install
```

Then open the `script-hts-ft.js` file in a code editor, such as VS Code.

Write the script

An almost-complete script has already been prepared for you, and you will only need to make a few modifications (outlined below) for it to run successfully.

Step 1: Configure HTS token to be created

To create a new HTS token, we will use `TokenCreateTransaction`. This transaction requires many properties to be set on it.

For fungible tokens (which are analogous to ERC20 tokens), set the token type to `TokenType.FungibleCommon`.

Set the token name and token symbol based on your name (or nickname).

Set the decimal property to 2.

Set the initial supply to 1 million.

```
{% code title="script-hts-ft.js" overflow="wrap" %}
js
```

```
    .setTokenType(TokenType.FungibleCommon)
    .setTokenName("bguiz coin")
    .setTokenSymbol("BGZ")
    .setDecimals(2)
    .setInitialSupply(1000000)
```

```
{% endcode %}
```

<details>

<summary>Key terminology for HTS token create transaction</summary>

Token Type: Fungible tokens, declared using `TokenType.FungibleCommon`, may be thought of as analogous to ERC20 tokens. Note that HTS also supports another token type, `TokenType.NonFungibleUnique`, which may be thought of as analogous to ERC721 tokens.

Token Name: This is the full name of the token. For example, "Singapore Dollar".

Token Symbol: This is the abbreviation of the token's name. For example, "SGD".

Decimals: This is the number of decimal places the currency uses. For example, 2 mimics "cents", where the smallest unit of the token is 0.01 (1/100) of a single token.

Initial Supply: This is the number of units of the token to "mint" when first creating the token. Note that this is specified in the smallest units, so 10000000 initial supply when decimals is 2, results in 10000 full units of the token being minted. It might be easier to think about it as "one million cents equals ten thousand dollars".

Treasury Account ID: This is the account for which the initial supply is credited. For example, using accountId would mean that your own account receives all the tokens when they are minted.

Admin Key: This is the account that is authorized to administrate this token. For example, using accountKey would mean that your own account would get to perform actions such as minting additional supply.

</details>

Step 2: Mirror Node API to query the specified token balance

Now, query the token balance of our account. Since the treasury account was configured as your own account, it will have the entire initial supply of the token.

You will want to use the Mirror Node API with the path
/api/v1/accounts/{idOrAliasOrEvmAddress}/tokens for this task.

Specify accountId within the URL path

Specify tokenId as the token.id query parameter

Specify 1 as the limit query parameter (you are only interested in one token)

Using string interpolation, construct accountBalanceFetchApiUrl like so:

```
{% code title="script-hts-ft.js" overflow="wrap" %}
js
  const accountBalanceFetchApiUrl =
  https://testnet.mirrornode.hedera.com/api/v1/accounts/${accountId}/tokens?
  token.id=${tokenId}&limit=1&order=desc;
```

```
{% endcode %}
```

<details>

<summary>Learn more about Mirror Node APIs</summary>

You can explore the Mirror Node APIs interactively via its Swagger page: Hedera Testnet Mirror Node REST API.

You can perform the same Mirror Node API query as accountBalanceFetchApiUrl above. This is what the relevant part of the Swagger page would look like when doing so:

```

```

You can learn more about the Mirror Nodes via its documentation: REST API.

</details>

Run the script

In the terminal, run the script using the following command:

```
shell
node script-hts-ft.js
```

You should see output similar to the following:

```
accountId: 0.0.1201
tokenId: 0.0.5878530
tokenExplorerUrl: https://hashscan.io/testnet/token/0.0.5878530
accountTokenBalance: 1000000
accountBalanceFetchApiUrl:
https://testnet.mirrornode.hedera.com/api/v1/accounts/0.0.1201/tokens?
token.id=0.0.5878530&limit=1&order=desc
```

Open tokenExplorerUrl in your browser and check that:

```

```

The token should exist, and its "token ID" should match tokenId. (1)
The "name" and "symbol" should be shown as the same values derived from your
name (or nickname) that you chose earlier. (2)
The "treasury account" should match accountId. (3)
Both the "total supply" and "initial supply" should be 10,000. (4)

```
{% hint style="info" %}
Note: "total supply" and "initial supply" are not displayed as 1,000,000 because
of the two decimal places configured. Instead, these are displayed as 10,000.00.
{% endhint %}
```

Complete

Congratulations, you have completed the Hedera Token Service Hello World
sequence! 🎉🎉🎉

You have learned how to:

- [x] Create and mint a new fungible token on HTS
- [x] Query the token balance

Next Steps

Now that you have completed this Hello World sequence, you have interacted with
Hedera Token Service (HTS). There are other Hello World sequences for Hedera
Smart Contract Service (HSCS), and Hedera File Service (HFS), which you may wish
to check out next.

Cheat sheet

```
<details>
```

```
<summary>Skip to final state</summary>
```

The repo, github.com/hedera-dev/hello-future-world, is intended to be used alongside this tutorial.

To skip ahead to the final state, use the completed branch. You may use this to compare your implementation to the completed steps of the tutorial.

```
shell
git fetch origin completed:completed
git checkout completed
```

Alternatively, you may view the completed branch on Github: github.com/hedera-dev/hello-future-world/tree/completed/04-hts-ft-sdk

</details>

Writer: Brendan Editors: Abi, Michiel, Ryan, Krystal

README.md:

Hello World


Hedera Quickstart Guide

So, you want to develop on Hedera but are unsure where to start? Why not start with something you could complete in 10 minutes? This quick start guide is designed to kickstart your Hedera development journey.

Let's begin!

Hello World Sequences

There are multiple Hello World sequences for you to follow along. To start, you must complete the first sequence, as you will need a funded account to do any other tasks on Hedera.

<p>START HERE</p> <p> Create &#x26; fund account</p>	create-fund-account.md	create-fund-account.md

Subsequently, you can complete the remaining sequences in any order.

<p>01. HFS: Files</p> <p> hfs-files</p>	hfs-files	hfs-files
<p>02. HTS: Fungible Token</p> <p> hts-fungible-token</p>	hts-fungible-token	hts-fungible-token
<p>03. HSCS: Smart Contract</p>	hscs-smart-contract	hscs-smart-contract

contract.md">HSCS: Smart Contract</td><td>smart-contracts-icon.png</td><td>hscs-smart-contract.md</td></tr><tr><td align="center">04.HCS: Topic</td><td>write-verifiable-data.png</td><td>hcs-topic.md</td></tr></tbody></table>

Core concepts

- Accounts & HBAR
- Mirror Node
- Transaction and Queries
- Hedera File Service (HFS)
- Hedera Token Service (HTS)
- Hedera Smart Contract Service (HSCS)
- Ethereum Virtual Machine (EVM)

Hello World Series Contributors

<p>Writer: Brendan, DevRel Engineer</p> <p>GitHub Blog</p>			
<p>Editor: Abi Castro, DevRel Engineer</p> <p>GitHub Twitter</p>			
<p>Editor: Michiel, Developer Advocate</p> <p>GitHub LinkedIn</p>			
<p>Editor: Ryan Arndt, DevRel Education</p> <p>GitHub LinkedIn</p>			

how-to-set-up-a-hedera-local-node.md:

How to Set Up a Hedera Local Node


The Hedera Local Node project enables developers to establish their own local network for development and testing. The local network comprises the consensus node, mirror node, JSON-RPC relay, and other Hedera products, and can be set up using the CLI tool and Docker. This setup allows you to seamlessly build and deploy smart contracts from your local environment.

By the end of this tutorial, you'll be equipped to run a Hedera local node and generate keys, allowing you to test your projects and deploy projects in your local environment.

Prerequisites

Node.js >= v14.x
NPM >= v6.14.17\\
Minimum 16GB RAM
Docker >= v20.10.x
Docker Compose >= v2.12.3
Have Docker running on your machine with the correct configurations.

<details>

<summary>Docker configuration 

Ensure the VirtioFS file sharing implementation is enabled in the docker settings.

.png)

Ensure the following configurations are set at minimum in Docker Settings -> Resources and are available for use:

CPUs: 6
Memory: 8GB
Swap: 1 GB
Disk Image Size: 64 GB

Ensure the Allow the default Docker sockets to be used (requires password) is enabled in Docker Settings -> Advanced.

Note: The image may look different if you are on a different version

</details>

\\Local node can be run using Docker or NPM but we will use Docker for this tutorial. Here are the installation steps for NPM.

Table of Contents

1. Start Your Local Network
2. Generate Keys
3. Stop Your Local Network
4. Additional Resources

Start Your Local Network

Open a new terminal and navigate to your preferred directory where your Hedera Local Node project will live. Run the following command to clone the repo and install dependencies to your local machine:

```
bash
git clone https://github.com/hashgraph/hedera-local-node.git
cd hedera-local-node
npm install
```

For Windows users: You will need to update the file endings of

compose-network/mirror-node/init.sh by running this in WSL:

```
bash
dos2unix compose-network/mirror-node/init.sh
```

Ensure Docker is installed and open on your machine before running this command to get the network up and running:

```
bash
// starts and generates the first 30 accounts
npm run start -- -d
```

or

```
// will start local node but will not generate the first 30 accounts
docker compose up -d
```

Generate Keys

To generate accounts with random private keys, run the generate-accounts command. Specify the number of accounts generated by appending the number to the hedera generate-account command. For example, to generate 5 accounts, run hedera generate-accounts 5.

<details>

<summary><code>hedera generate-accounts 5</code> </summary>

Generating accounts in synchronous mode...

```
|-----|
|-----|
|-----| Accounts list ( ECDSA  keys)
|-----|
|-----|
|-----|
| id | private key |
balance |
|-----|
|-----|
| 0.0.1033 - 0xcd34a00d3fff542e350a5e61cb41509812bf23ea581f83a0a862c94d8c69704
- 10000 h |
| 0.0.1034 - 0xa4189ab682ba43925ce654ca09800bba86cf8b1b7f889006d5170d95f4fed365
- 10000 h |
| 0.0.1035 - 0xf9106e9841677136c9cbe8c114dab80470ca62a15bfe9c777006bcb114288c22
- 10000 h |
| 0.0.1036 - 0xe3517a9235971be1e1f95e791f3ffd7d753a652799fa11f1ace626036c4db275
- 10000 h |
| 0.0.1037 - 0x636926cf2f6f9fd0a58043c600390eeef0bbed9d4b8a113ea68a8d67f922d04e
- 10000 h |
|-----|
|-----|
|-----|
|-----| Accounts list (Alias ECDSA
keys) |-----|
|-----|
|-----|
| id | public address |
```

```

private key                                | balance |
|-----|-----|
| 0.0.1038 - 0xaBE90e20f394629e054Bc1E8F1338Fe8ea94F0b5 -
0x444913bd258f764e62db6c87abde7ca52ec22985db8c91b8c3b2b4f2c51775f0 - 10000 ¢ |
| 0.0.1039 - 0x26d941d8E1f6bF9B0F7e5156fA6ff02acEd0DF3E -
0xea25f427caf7029989669f93926b7902dde5361b176b4bc17b8ec0a967beaa0b - 10000 ¢ |
| 0.0.1040 - 0x64001c2d1f3a8d3574435B4F125944018E2E584D -
0xf2deb678a1e67e288d8a128334f41c890e7600b2a5471ecc9a3af4824e3021b7 - 10000 ¢ |
| 0.0.1041 - 0x6bE22CD9D16b64969683B74897E4EBB30c7c30E8 -
0xb9c2480cdbdddb2ecd6e032b87820c29e8791ad4f53b89f829269d856c835819 - 10000 ¢ |
| 0.0.1042 - 0x992d8aD211b28B23589c0b3Fe30de6C90662C4aB -
0x7e8bb0d85a8d80fa2eb2c9f6bd5c9b1a2c2f9f6992c7fffd201c8e81f0ec0000 - 10000 ¢ |
|-----|-----|

|-----|
|-----|
|-----| Accounts list (ED25519 keys)
|-----|
|-----|
|-----|
| id | private key |
balance |
|-----|
| 0.0.1043 - 0xd4917e152ca922b8bfba9fffc3486512ae25ec0a75b05c44f517b11cd12fd949b
- 10000 ¢ |
| 0.0.1044 - 0xbaeec69382fbb43e4d521b3d8717c9cba610a1fbcaededaaf4408c3138a683ae
- 10000 ¢ |
| 0.0.1045 - 0x1f5c4b2efd3c36d29e9d2e16a825abd001f99bff2388bb8c6011cd5f956023c9
- 10000 ¢ |
| 0.0.1046 - 0x1976acdd5e71ce7e8db4cb0aa112fa1c16876155f0f20b9b7029916073f1d67f
- 10000 ¢ |
| 0.0.1047 - 0x6e29f48b11ffc77e277f0500d607b35956da58f1ed30aad003fb1846bffffc483
- 10000 ¢ |
|-----|
|-----|

```

</details>

{% hint style="info" %}

Please note: Since the first 10 accounts generated are with predefined private keys, if you need 5 generated with random keys, you will run hedera start 15. The same rule applies when you use the hedera generate-accounts command.

{% endhint %}

Grab any of the account private keys generated from the Alias ECDSA keys Accounts list. This will be used as the LOCALNODEOPERATORPRIVATEKEY environment variable value in your .env file of your project.

Stop Your Local Network

To stop your local node, you can run the hedera stop command. If you want to keep any files created manually in the working directory, please save them before executing this command.

<details>

<summary><code>hedera stop</code></summary>

Stopping the network...
Stopping the docker containers...
Cleaning the volumes and temp files...

</details>

Alternatively, run `docker compose down -v; git clean -xfd; git reset --hard` to stop the local node and reset it to its original state.

<details>

<summary><code>docker compose down -v; git clean -xfd; git reset --hard</code></summary>

bash

[+] Running 27/27

✓ Container mirror-node-web3	Removed	3.5s
✓ Container json-rpc-relay-ws	Removed	10.8s
✓ Container mirror-node-monitor	Removed	3.7s
✓ Container relay-cache	Removed	0.9s
✓ Container prometheus	Removed	0.9s
✓ Container record-sidecar-uploader	Removed	0.0s
✓ Container grafana	Removed	0.9s
✓ Container hedera-explorer	Removed	10.4s
✓ Container json-rpc-relay	Removed	10.7s
✓ Container account-balances-uploader	Removed	0.1s
✓ Container envoy-proxy	Removed	1.0s
✓ Container mirror-node-grpc	Removed	2.7s
✓ Container mirror-node-rest	Removed	10.4s
✓ Container network-node	Removed	10.8s
✓ Container mirror-node-importer	Removed	10.4s
✓ Container record-streams-uploader	Removed	0.0s
✓ Container haveged	Removed	0.0s
✓ Container mirror-node-db	Removed	0.3s
✓ Container minio	Removed	0.0s
✓ Volume prometheus-data	Removed	0.0s
✓ Volume minio-data	Removed	0.0s
✓ Volume mirror-node-postgres	Removed	0.1s
✓ Volume grafana-data	Removed	0.2s
✓ Network network-node-bridge	Removed	0.1s
✓ Network hedera-local-nodedefault	Removed	0.2s
✓ Network cloud-storage	Removed	0.2s
✓ Network mirror-node	Removed	0.2s

Removing .husky//

Removing network-logs/




Removing nodemodules/

HEAD is now at

</details>

 Note: All available commands can be checked out [here](#).

Additional Resources

-  [Hedera Local Node Repository](#)
-  [Hedera Local Node CLI Tool Commands](#)
-  [Hedera Local Node Docker Setup \\[Video Tutorial\]](#)

README.md:

Local Node

setup-hedera-node-cli-npm.md:

Set Up a Hedera Local Node using the NPM CLI

Hedera is an open-source, public, proof-of-stake network. Its network services offer low and fixed fees, 10k TPS, and instant transaction finality. Learn more about the Hedera platform and how it works.

In this tutorial, we will adopt, set up, and run a Hedera node locally using the @hashgraph/hedera-local NPM Command Line Interface (CLI) tool with docker compose.

> This tutorial is based on the Hedera Local Node README documentation.

> Already familiar with using a cloud service? Check out the other options for setting up and running the Hedera node locally. See the Useful resources section for more information.

Prerequisites

To get started with this tutorial, ensure that you have the following software installed:

Node.js >= v14.x (Check version: node -v)
NPM >= v6.14.17 (Check version: npm -v)
Docker >= v20.10.x (Check version: docker -v)
Docker Compose >= v2.12.3 (Check version: docker compose version)
Hardware: Minimum 16GB RAM

Installation

Node.js and NPM: Refer to the official installation guide.
Docker: See Docker Setup Guide to get docker up and running (note: specific instructions may vary based on the OS).

Getting Started

Clone the GitHub repo, navigate to the project folder using the commands below;

```
js
git clone https://github.com/hashgraph/hedera-local-node.git
cd hedera-local-node
```

Install CLI Tool

The command below can be used to install the official release from the NPM repository.

```
js
npm install @hashgraph/hedera-local -g
```

> Note: This version may not reflect the most recent changes to the main branch of this repository. It also uses a baked in version of the Docker Compose definitions and will not reflect any local changes made to the repository.

Local development Installation

Install the dependencies locally.

```
js
npm install && npm install -g
```

Running the Node:

Start the local node (Note: Ensure Docker is running):

```
js
npm run start
```

You can pass the following CLI flags, this would be used later in the following sections:

```
js
--d / --detached - Start the local node in detached mode.
--h / --host - Override the default host.
```

Other NPM commands:

```
npm run restart to restart the network
npm run stop to stop the network
npm run generate-accounts to generate new accounts - network must be running
first
```

You should see the following response in the terminal:

```
bash
hedera-local-node % npm run start
```

```
> @hashgraph/hedera-local@2.26.2 restart
> npm run build && node ./build/index.js restart
```

```
> @hashgraph/hedera-local@2.26.2 build
> rimraf ./build && tsc
```

```
[Hedera-Local-Node] INFO (StateController) [✓] Starting restart procedure!
[Hedera-Local-Node] INFO (CleanUpState) ⌚ Initiating clean up procedure. Trying
to revert unneeded changes to files...
[Hedera-Local-Node] INFO (CleanUpState) [✓] Clean up of consensus node
properties finished.
[Hedera-Local-Node] INFO (CleanUpState) [✓] Clean up of mirror node properties
finished.
[Hedera-Local-Node] INFO (StopState) ⌚ Initiating stop procedure. Trying to
stop docker containers and clean up volumes...
[Hedera-Local-Node] INFO (StopState) ⌚ Stopping the network...
[Hedera-Local-Node] INFO (StopState) [✓] Hedera Local Node was stopped
successfully.
[Hedera-Local-Node] INFO (InitState) ⌚ Making sure that Docker is started and
it is correct version...
[Hedera-Local-Node] INFO (DockerService) ⌚ Checking docker compose version...
[Hedera-Local-Node] INFO (DockerService) ⌚ Checking docker resources...
[Hedera-Local-Node] WARNING (DockerService) [!] Port 3000 is in use.
[Hedera-Local-Node] INFO (InitState) ⌚ Setting configuration with latest images
on host 127.0.0.1 with dev mode turned off using turbo mode in single node
configuration...
```

```
[Hedera-Local-Node] INFO (InitState) [✓] Local Node Working directory set to
/Users/owanate/Library/Application Support/hedera-local.
[Hedera-Local-Node] INFO (InitState) [✓] Hedera JSON-RPC Relay rate limits were
disabled.
[Hedera-Local-Node] INFO (InitState) [✓] Needed environment variables were set
for this configuration.
[Hedera-Local-Node] INFO (InitState) [✓] Needed bootstrap properties were set
for this configuration.
[Hedera-Local-Node] INFO (InitState) [✓] Needed bootstrap properties were set
for this configuration.
[Hedera-Local-Node] INFO (InitState) [✓] Needed mirror node properties were set
for this configuration.
[Hedera-Local-Node] INFO (StartState) ⌚ Starting Hedera Local Node...
```

To generate default accounts and start the local node in detached mode, use the command below:

```
js
npm run start -- -d
```

You should see the following response in the terminal:

```
bash
hedera-local-node % npm run start -- -d

> @hashgraph/hedera-local@2.26.2 start
> npm run build && node ./build/index.js start -d

> @hashgraph/hedera-local@2.26.2 build
> rimraf ./build && tsc
[Hedera-Local-Node] INFO (StartState) [✓] Hedera Local Node successfully
started!
[Hedera-Local-Node] INFO (NetworkPrepState) ⌚ Starting Network Preparation
State...
[Hedera-Local-Node] INFO (NetworkPrepState) [✓] Imported fees successfully!
[Hedera-Local-Node] INFO (NetworkPrepState) [✓] Topic was created!
[Hedera-Local-Node] INFO (AccountCreationState) ⌚ Starting Account Creation
state in synchronous mode ...
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) |-----|
Accounts list (ECDSA keys) |-----|
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) |    id    |
private key                | balance |
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1002 -
0x7f109a9e3b0d8ecfba9cc23a3614433ce0fa7ddcc80f2a8f10b222179a5a80d6 - 10000 ¢ |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1003 -
0x6ec1f2e7d126a74a1d2ff9e1c5d90b92378c725e506651ff8bb8616a5c724628 - 10000 ¢ |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1004 -
0xb4d7f7e82f61d81c95985771b8abf518f9328d019c36849d4214b5f995d13814 - 10000 ¢ |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1005 -
0x941536648ac10d5734973e94df413c17809d6cc5e24cd11e947e685acfbfd12ae - 10000 ¢ |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1006 -
0x5829cf333ef66b6bdd34950f096cb24e06ef041c5f63e577b4f3362309125863 - 10000 ¢ |
```

```

[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1007 -
0x8fc4bffe2b40b2b7db7fd937736c4575a0925511d7a0a2dfc3274e8c17b41d20 - 10000 h |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1008 -
0xb6c10e2baaeba1fa4a8b73644db4f28f4bf0912cceb6e8959f73bb423c33bd84 - 10000 h |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1009 -
0xfe8875acb38f684b2025d5472445b8e4745705a9e7adc9b0485a05df790df700 - 10000 h |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1010 -
0xbdc6e0a69f2921a78e9af930111334a41d3fab44653c8de0775572c526feea2d - 10000 h |
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1011 -
0x3e215c3d2a59626a669ed04ec1700f36c05c9b216e592f58bbfd3d8aa6ea25f9 - 10000 h |
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
[Hedera-Local-Node] INFO (AccountCreationState)
|-----| Accounts list (Alias ECDSA
keys) |-----|
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) | id |
public address | private key
| balance |
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) | 0.0.1012 -
0x67d8d32e9bf1a9968a5ff53b87d777aa8ebbee69 -
0x105d050185ccb907fba04dd92d8de9e32c18305e097ab41dadda21489a211524 - 10000 h |
.....
[Hedera-Local-Node] INFO (AccountCreationState)
|-----|
[Hedera-Local-Node] INFO (AccountCreationState) [✓] Accounts created
successfully!
[Hedera-Local-Node] INFO (CleanUpState) ⌚ Initiating clean up procedure. Trying
to revert unneeded changes to files...
[Hedera-Local-Node] INFO (CleanUpState) [✓] Clean up of consensus node
properties finished.
[Hedera-Local-Node] INFO (CleanUpState) [✓] Clean up of mirror node properties
finished.

```

!Running Hedera Node on Terminal

Verify Running Node

There are different ways to verify that a node is running;

Check Block Number using Hashscan Block Explorer
Send cURL request to getBlockNumber

Check Block Number using Hashscan Block Explorer

Visit the local mirror node explorer endpoint
(<http://localhost:8080/devnet/dashboard>) in your web browser. Ensure that
LOCALNET is selected, as this will show you the Hedera network running within
your local network.

Select any of the listed blocks to view the details (Consensus, Block,
Transaction Hash, etc) for a particular block.


```
> @hashgraph/hedera-local@2.26.2 build
> rimraf ./build && tsc
```

```
[Hedera-Local-Node] INFO (StateController) [✓] Starting start procedure!
[Hedera-Local-Node] INFO (InitState) ⌚ Making sure that Docker is started and
it is correct version...
[Hedera-Local-Node] INFO (DockerService) ⌚ Checking docker compose version...
[Hedera-Local-Node] INFO (DockerService) ⌚ Checking docker resources...
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 5551 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 8545 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 5600 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 5433 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 8082 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Port 6379 is in use.
[Hedera-Local-Node] WARNING (DockerService) [!] Port 7546 is in use.
[Hedera-Local-Node] WARNING (DockerService) [!] Port 8080 is in use.
[Hedera-Local-Node] WARNING (DockerService) [!] Port 3000 is in use.
[Hedera-Local-Node] ERROR (DockerService) [✗] [✗] Node cannot start properly
because necessary ports are in use!
```

Fix

Option 1: Instead of starting another instance of the network, use the npm run generate-accounts to generate new accounts for an already started network.

Option 2: If you get the above error, ensure that you terminate any existing Docker processes for the local node, and also any other processes that are bound to these port numbers, before running the npm start command. You can run docker compose down -v, git clean -xfd, git reset --hard to fix this.

Useful Terms

For an in depth explanation of the different terms below, see the glossary documentation.

- Accounts list (ED25519 keys)
- Private keys
- Public address

Next Steps

Want to learn how to deploy smart contracts on Hedera? Visit the guide on how to Deploy a Smart Contract Using Hardhat and Hedera JSON-RPC Relay.

Summary

In this tutorial, we successfully set up and ran the Hedera local node using the NPM CLI tool, generated default accounts and solved common errors encountered when running the local node.

Useful Resources

- Set and Run a Hedera Node using the Local Hedera Package.
- Setup node using Docker CLI.
- Use local network variables to interact with Consensus and Mirror Nodes
- Using Grafana and Prometheus Endpoints.

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Owanate, Technical Writer</p><p><a
href="https://github.com/owans">GitHub</a> | <a
href="https://https/medium.com/@owanateamachree">Medium</a></p></td><td><a
href="https://medium.com/@owanateamachree">https://medium.com/@owanateamachree</
```

GitHub Twitter	https://twitter.com/theekrystallee
---	---

codespaces.md:

Run a Local Node in Codespaces

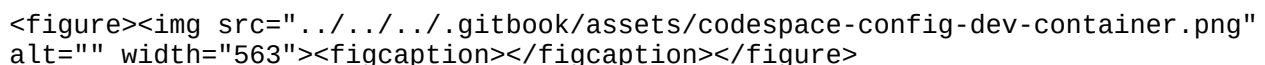
Codespaces is a cloud development environment (CDE) that's hosted in the cloud. You can customize your project for GitHub Codespaces by committing configuration files to your repository (often known as Configuration-as-Code), which creates a repeatable codespaces configuration for all users of your project. [GitHub Codespaces overview](#)

Prerequisites

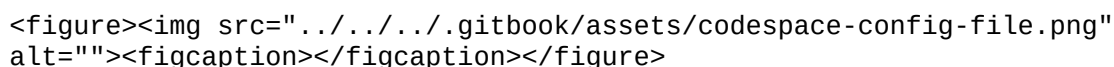
Review first the Quickstart for GitHub Codespaces guide.
 Install VS Code Desktop application.
 In Editor preference change your client to Visual Studio Code (Should not be Visual Studio Code for the Web)

Configure Dev Container

To configure the dev container, open the Hedela Local Node repo and click on the Code->Codespaces->...-> Configure dev container.



This will open the dev container configuration file where you can customize your configuration like the CPUs and memory.



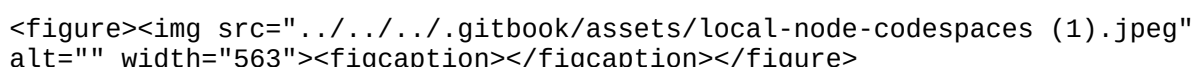
{% hint style="info" %}

Note: If you make changes to your config file, commit and push your changes before running local node, to ensure the project starts with the right configuration.

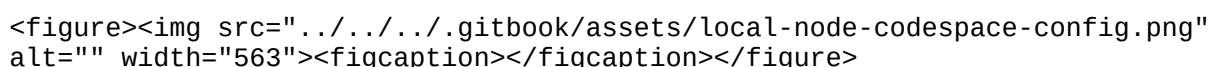
{% endhint %}

Creating and Running Your Codespace

Open the Hedela Local Node repo and click on the Code->Codespaces->...-> New with options... button and choose the appropriate settings:



Once your codespace is created, the template repository will be automatically cloned into it. Your codespace is all set up and have the local node running!



Conclusion and Additional Resources

Congrats on successfully setting up your Codespace and running a Hedera Local Node!

- ☞ Hedera Local Node Repository
- ☞ Quickstart for GitHub Codespaces
- ☞ Adding Dev Container Config to Repo

gitpod.md:

Run a Local Node in Gitpod

The local network comprises the consensus node, mirror node, JSON-RPC relay, and other Hedera services and now be set up without Docker and draining your computer's resources by using Gitpod. Gitpod provides Cloud Development Environments (CDEs) and allows developers to work from any device without the need to maintain static and brittle local development environments. By the end of this tutorial, you will have your Hedera local node running on Gitpod.

Prerequisites

Signed into your GitHub account in your browser.
Register a Gitpod account with your GitHub account.
If this is your first time using Gitpod, please read the Gitpod getting started guide.
Install the browser extension: Gitpod browser extension.
The Mirror Node Web Explorer requires VS Code Desktop to be installed, as VS Code Browser has limitations related to communicating with local ports, e.g. <http://127.0.0.1:5551/>.

Set Up Gitpod Permissions

Enable publicrepo permission for GitHub provider on Gitpod's Git integrations page.

<figure><figcaption></figcaption></figure>

<figure><figcaption></figcaption></figure>

Running the Hedera Local Node

The hedera-local-node project repository already has a Gitpod configuration file (.gitpod.yml), which makes it easy to run it within a workspace on Gitpod. Open the Hedera Local Node repo. Click on the Gitpod Open button.

<figure><figcaption></figcaption></figure>

The Gitpod browser extension modifies the Github UI to add this button. This will spin up a new Gitpod workspace with your choice of CDE which will run the

Hedera Local Node in your cloud environment.

Testing the Setup

To confirm everything is running smoothly, run the curl commands below to query the mirror node for a list of accounts, query the JSON-RPC relay for the latest block, and open the mirror node explorer (HashScan) using the local endpoint (<http://localhost:8080/devnet/dashboard>).

Mirror Node REST API

The following command queries the Mirror Node for a list of accounts on your Hedera network.

```
bash
curl "http://localhost:5551/api/v1/accounts" \
  -X GET
```

See the Mirror Node interact API docs for a full list of available APIs.

JSON RPC Relay

The following command queries the RPC Relay for the latest block on your Hedera network.

```
bash
curl "<http://localhost:7546>" \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{"method":"ethgetBlockByNumber","params":
["latest",false,"id":1,"jsonrpc":"2.0"]}'
```

See the endpoint table in [hedera-json-rpc-relay](#) for a full list of available RPCs.

Mirror Node Explorer (Hashscan)

Visit the local mirror node explorer endpoint (<http://localhost:8080/devnet/dashboard>) in your web browser. Ensure that LOCALNET is selected, as this will show you the Hedera network running within your Gitpod, and not one of the public nodes.

<figure><figcaption></figcaption></figure>

Shut Down the Gitpod Workspace

{% hint style="warning" %}

Note: Gitpod usage is billed by the hour on paid plans, and hours are limited on the free plans. Therefore, once completed, remember to stop the Gitpod workspace.

{% endhint %}

<figure><figcaption></figcaption></figure>

Conclusion and Additional Resources

Congrats on successfully setting up your Gitpod workspace and running a Hedera Local Node!

[→ Hedera Local Node Repository](#)

[→ Gitpod Documentation](#)

README.md:

How to Run Hedera Local Node in a Cloud Development Environment (CDE)

The Hedera Local Node project enables developers to establish their own local network for development and testing. The local network comprises the consensus node, mirror node, JSON-RPC relay, and other Hedera services be set up without Docker and draining your computer's resources by using .Cloud Development Environments (CDEs). CDEs allows developers to work from any device without the need to maintain static and brittle local development environments.

Available Services

The Hedera local node comes with various services, each serving different functions, and accessible locally. These are the endpoints for each service:

Type	Endpoint

Consensus Node Endpoint	http://localhost:50211/
Mirror Node GRPC Endpoint	http://localhost:5600/
Mirror Node REST API Endpoint	http://localhost:5551/
JSON RPC Relay Endpoint	http://localhost:7546/
JSON RPC Relay Websocket Endpoint	http://localhost:8546/
Mirror Node Explorer (HashScan)	http://localhost:8080/devnet/dashboard
Grafana UI	http://localhost:3000/
Prometheus UI	http://localhost:9090/

You may access these services on localhost, and these endpoints are set up to be accessed from your own computer as if they were running locally. Since Gitpod and Codespaces are cloud-based development environments, "localhost" here refers to a virtual environment on cloud servers that you're accessing through your browser. Gitpod and Codespaces redirects these local addresses to your cloud workspace, making it feel as though you're working on a local setup.

develop-a-hedera-dapp-integrated-with-walletconnect.md:

description: >-

In the dynamic world of decentralized applications (DApps), catering to users with diverse wallet preferences is important.

Build a Hedera DApp integrated with walletconnect

Explore DApp development using the Mirror Node API and Hedera Token Service (HTS). Discover how to integrate HTS functionality into your DApp for seamless token management and transactions. This guide uses React, Material UI, Ethers,

and TypeScript with the Create React App (CRA) Hedera DApp template integrated with walletconnect, streamlining your development process.

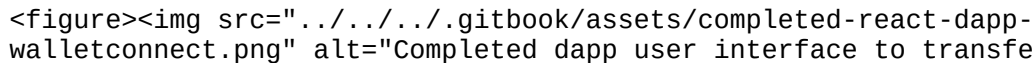
What you will accomplish

- [] Query the mirror node for account token balance, token information, and Non-fungible information.

- [] Query the mirror node to check if the receiver has associated with a token ID

- [] Associate an HTS token with HashPack, Kabila, Blade, or MetaMask through a UI

- [] Transfer an HTS token through a UI


Completed DAPP User Interface

Prerequisites

Before you begin, you should be familiar with the following:

- JavaScript
- TypeScript
- React
- REST API

<details>


<summary>Also, you should have the following set up on your computer 

</summary>

- [x] git installed
 - Minimum version: 2.37
 - Recommended: Install Git (Github)
- [x] A code editor or IDE
 - Recommended: VS Code. Install VS Code (Visual Studio)
- [x] NodeJs + npm installed
 - Minimum version of NodeJs: 18
 - Minimum version of npm: 9.5
 - Recommended for Linux & Mac: nvm
 - Recommended for Windows: nvm-windows

</details>

<details>

<summary>Check your prerequisites set up  </summary>

Open your terminal, and enter the following commands.

```
shell
git --version
code --version
node --version
npm --version
```

Each of these commands should output some text that includes a version number, for example:

```
text
git --version
git version 2.39.2 (Apple Git-143)
```

```
code --version
1.81.1
6c3e3dba23e8fadc360aed75ce363ba185c49794
arm64
```

```
node --version
v20.6.1
```

```
npm --version
9.8.1
```

If the output contains text similar to command not found, please install that item.

</details>

Get Started

We choose to scaffold our project by using the CRA Hedera DApp template, as it offers:

- [x] Multi-wallet integration via walletconnect supporting MetaMask and Hedera native wallets
- [x] Mirror Node Client
- [x] State management via React Context
- [x] Material UI
- [x] Choice of TypeScript or JavaScript

This custom template eliminates setup overhead and allows you to dive straight into the core features of your project.

<details>

<summary>Template Project Repos</summary>

The complete TypeScript project can be found on GitHub [here](#).

The complete JavaScript project can be found on GitHub [here](#).

</details>

1. Scaffold your project

Open a terminal and run the following command to set up your project structure, replacing my-app-name with your desired directory name.

```
shell
npx create-react-app <my-app-name> --template git+ssh://git@github.com/hedera-dev/cra-hedera-dapp-template.git
```

<details>

<summary>Scaffolding project expected output</summary>

<figure><figcaption><p>Scaffold project expected output</p></figcaption></figure>

</details>

Open your newly created react app project with visual studio code. You should see the following file structure.

<figure><figcaption><p>Hedera React DApp File Structure</p></figcaption></figure>

2. Fetching Token Data: Writing Mirror Node API Queries

Mirror nodes offer access to historical data from the Hedera network while optimizing the use of network resources. You can easily retrieve information like transactions, records, events, and balances. Visit the mirror node API docs to learn more.

In vscode open the file located at `src/services/wallets/mirrorNodeClient.ts`.

This file creates a mirror node client and is used to fetch data from the mirror nodes. We will add new code to help us obtain information about the tokens we currently own.

```
{% hint style="info" %}
```

This client is configured for the Hedera Testnet. For further configuration, go to `src/config/network.ts`.

```
{% endhint %}
```

2.1 Query Account Token Balances by Account ID

We'll use the Mirror Node API to query information about the tokens we currently own and the quantities of those tokens.

Open `src/services/wallets/mirrorNodeClient.ts` and paste the below interface outside of and above the `MirrorNodeClient` class.

Typescript

```
export interface MirrorNodeAccountTokenBalance {  
  balance: number,  
  tokenId: string,  
}
```

```
{% hint style="info" %}
```

This interface defines the data fields we need for our DApp, filtering out any extra data from the mirror node response.

```
{% endhint %}
```

Paste the below HTTP GET request outside of and below the `MirrorNodeClient` class in the `src/services/wallets/mirrorNodeClient.ts` file.

Typescript

```
// Purpose: get token balances for an account  
// Returns: an array of MirrorNodeAccountTokenBalance  
async getAccountTokenBalances(accountId: AccountId) {  
  // get token balances  
  const tokenBalanceInfo = await fetch(`${this.url}/api/v1/accounts/${  
accountId}/tokens?limit=100, { method: "GET" });  
  const tokenBalanceInfoJson = await tokenBalanceInfo.json();  
  
  const tokenBalances = [...tokenBalanceInfoJson.tokens] as  
MirrorNodeAccountTokenBalance[];
```

```
  // because the mirror node API paginates results, we need to check if there  
  are more results
```

```
  // if links.next is not null, then there are more results and we need to
```

```

fetch them until links.next is null
    let nextLink = tokenBalanceInfoJson.links.next;
    while (nextLink !== null) {
        const nextTokenBalanceInfo = await fetch(`${this.url}${nextLink}`, { method:
"GET" });
        const nextTokenBalanceInfoJson = await nextTokenBalanceInfo.json();
        tokenBalances.push(...nextTokenBalanceInfoJson.tokens);
        nextLink = nextTokenBalanceInfoJson.links.next;
    }

    return tokenBalances;
}

```

<details>

<summary>File Checkpoint</summary>

To ensure you're on the right track, your
src/services/wallets/mirrorNodeClient.ts file should look like below.

Typescript

```

import { AccountId } from "@hashgraph/sdk";
import { NetworkConfig } from "../../config";

export interface MirrorNodeAccountTokenBalance {
    balance: number,
    tokenId: string,
}

export class MirrorNodeClient {
    url: string;
    constructor(networkConfig: NetworkConfig) {
        this.url = networkConfig.mirrorNodeUrl;
    }

    // Purpose: get token balances for an account
    // Returns: an array of MirrorNodeAccountTokenBalance
    async getAccountTokenBalances(accountId: AccountId) {
        // get token balances
        const tokenBalanceInfo = await fetch(`${this.url}/api/v1/accounts/${
accountId}/tokens?limit=100, { method: "GET" });
        const tokenBalanceInfoJson = await tokenBalanceInfo.json();

        const tokenBalances = [...tokenBalanceInfoJson.tokens] as
MirrorNodeAccountTokenBalance[];

        // because the mirror node API paginates results, we need to check if there
are more results
        // if links.next is not null, then there are more results and we need to fetch
them until links.next is null
        let nextLink = tokenBalanceInfoJson.links.next;
        while (nextLink !== null) {
            const nextTokenBalanceInfo = await fetch(`${this.url}${nextLink}`, { method:
"GET" });
            const nextTokenBalanceInfoJson = await nextTokenBalanceInfo.json();
            tokenBalances.push(...nextTokenBalanceInfoJson.tokens);
            nextLink = nextTokenBalanceInfoJson.links.next;
        }

        return tokenBalances;
    }

    async getAccountInfo(accountId: AccountId) {
        const accountInfo = await fetch(`${this.url}/api/v1/accounts/${accountId},

```

```

{ method: "GET" });
    const accountInfoJson = await accountInfo.json();
    return accountInfoJson;
  }
}

```

</details>

2.2 Query Token Information by Token ID

In the previous step we wrote code to obtain the current token balance of an account. Next we will retrieve the type of token (Non-Fungible or Fungible), decimal precision, token name and symbol.

Open `src/services/wallets/mirrorNodeClient.ts` and paste the interface outside of and above the `MirrorNodeClient` class.

```

Typescript
export interface MirrorNodeTokenInfo {
  type: 'FUNGIBLECOMMON' | 'NONFUNGIBLEUNIQUE',
  decimals: string,
  name: string,
  symbol: string,
  tokenId: string,
}

```

Paste the below HTTP GET request outside of and below the `getAccountTokenBalances` function in the `src/services/wallets/mirrorNodeClient.ts` file.

```

Typescript
// Purpose: get token info for a token
// Returns: a MirrorNodeTokenInfo
async getTokenInfo(tokenId: string) {
  const tokenInfo = await fetch(`${this.url}/api/v1/tokens/${tokenId}`, { method:
"GET" });
  const tokenInfoJson = await tokenInfo.json() as MirrorNodeTokenInfo;
  return tokenInfoJson;
}

```

2.3 Query Account NFT Information by AccountID

In the previous step we wrote code to obtain the token details (token type, decimals, name, and symbol). Next we will retrieve the NFT serial numbers that are owned.

Open `src/services/wallets/mirrorNodeClient.ts` and paste the interface outside of and above the `MirrorNodeClient` class.

```

Typescript
export interface MirrorNodeNftInfo {
  tokenId: string,
  serialnumber: number,
}

```

Paste the below HTTP GET request outside of and below the `getTokenInfo` function in the `src/services/wallets/mirrorNodeClient.ts` file.

Typescript

```

// Purpose: get NFT Info for an account
// Returns: an array of NFTInfo
async getNftInfo(accountId: AccountId) {
  const nftInfo = await fetch(`${this.url}/api/v1/accounts/${accountId}/nfts?
limit=100, { method: "GET" });
  const nftInfoJson = await nftInfo.json();

  const nftInfos = [...nftInfoJson.nfts] as MirrorNodeNftInfo[];

  // because the mirror node API paginates results, we need to check if there
  are more results
  // if links.next is not null, then there are more results and we need to fetch
  them until links.next is null
  let nextLink = nftInfoJson.links.next;
  while (nextLink !== null) {
    const nextNftInfo = await fetch(`${this.url}${nextLink}`, { method: "GET" });
    const nextNftInfoJson = await nextNftInfo.json();
    nftInfos.push(...nextNftInfoJson.nfts);
    nextLink = nextNftInfoJson.links.next;
  }
  return nftInfos;
}

```

2.4 Combine Account Token Balances and Token Information via Data Aggregation

We need to combine all of our HTTP response data in order to display our available tokens in our DApp.

Open `src/services/wallets/mirrorNodeClient.ts` and paste the interface outside of and above the `MirrorNodeClient` class.

```

Typescript
export interface MirrorNodeAccountTokenBalanceWithInfo extends
MirrorNodeAccountTokenBalance {
  info: MirrorNodeTokenInfo,
  nftSerialNumbers?: number[],
}

```

Paste the function outside of and below the `getNftInfo` function in the `src/services/wallets/mirrorNodeClient.ts` file.

```

Typescript
// Purpose: get token balances for an account with token info in order to
display token balance, token type, decimals, etc.
// Returns: an array of MirrorNodeAccountTokenBalanceWithInfo
async getAccountTokenBalancesWithTokenInfo(accountId: AccountId):
Promise<MirrorNodeAccountTokenBalanceWithInfo[]> {
  //1. Retrieve all token balances in the account
  const tokens = await this.getAccountTokenBalances(accountId);
  //2. Create a map of token IDs to token info and fetch token info for each
  token
  const tokenInfos = new Map<string, MirrorNodeTokenInfo>();
  for (const token of tokens) {
    const tokenInfo = await this.getTokenInfo(token.tokenid);
    tokenInfos.set(tokenInfo.tokenid, tokenInfo);
  }

  //3. Fetch all NFT info in account
  const nftInfos = await this.getNftInfo(accountId);

  //4. Create a map of token IDs to arrays of serial numbers
  const tokenIdToSerialNumbers = new Map<string, number[]>();
}

```

```

for (const nftInfo of nftInfos) {
  const tokenId = nftInfo.tokenid;
  const serialNumber = nftInfo.serialnumber;

  // if we haven't seen this tokenId before, create a new array with the
  serial number
  if (!tokenIdToSerialNumbers.has(tokenId)) {
    tokenIdToSerialNumbers.set(tokenId, [serialNumber]);
  } else {
    // if we have seen this tokenId before, add the serial number to the array
    tokenIdToSerialNumbers.get(tokenId)!.push(serialNumber);
  }
}

//5. Combine token balances, token info, and NFT info and return
return tokens.map(token => {
  return {
    ...token,
    info: tokenInfos.get(token.tokenid)!,
    nftSerialNumbers: tokenIdToSerialNumbers.get(token.tokenid)
  }
});
}

```

{% hint style="info" %}

The `getAccountTokenBalancesWithTokenInfo` combines token balances, token info and, NFT info in order to display our available tokens in our DApp.

{% endhint %}

<details>

<summary>Complete `mirrorNodeClient.ts` file Checkpoint</summary>

To ensure you're on the right track, your `src/services/wallets/mirrorNodeClient.ts` file should look like below.

Typescript

```

import { AccountId } from "@hashgraph/sdk";
import { NetworkConfig } from "../../config";

```

```

export interface MirrorNodeAccountTokenBalance {
  balance: number,
  tokenId: string,
}

```

```

export interface MirrorNodeTokenInfo {
  type: 'FUNGIBLECOMMON' | 'NONFUNGIBLEUNIQUE',
  decimals: string,
  name: string,
  symbol: string,
  tokenId: string,
}

```

```

export interface MirrorNodeNftInfo {
  tokenId: string,
  serialnumber: number,
}

```

```

export interface MirrorNodeAccountTokenBalanceWithInfo extends
MirrorNodeAccountTokenBalance {
  info: MirrorNodeTokenInfo,
  nftSerialNumbers?: number[],
}

```



```

export class MirrorNodeClient {
  url: string;
  constructor(networkConfig: NetworkConfig) {
    this.url = networkConfig.mirrorNodeUrl;
  }

  // Purpose: get token balances for an account
  // Returns: an array of MirrorNodeAccountTokenBalance
  async getAccountTokenBalances(accountId: AccountId) {
    // get token balances
    const tokenBalanceInfo = await fetch(`${this.url}/api/v1/accounts/${accountId}/tokens?limit=100, { method: "GET" });
    const tokenBalanceInfoJson = await tokenBalanceInfo.json();

    const tokenBalances = [...tokenBalanceInfoJson.tokens] as MirrorNodeAccountTokenBalance[];

    // because the mirror node API paginates results, we need to check if there are more results
    // if links.next is not null, then there are more results and we need to fetch them until links.next is null
    let nextLink = tokenBalanceInfoJson.links.next;
    while (nextLink !== null) {
      const nextTokenBalanceInfo = await fetch(`${this.url}${nextLink}`, { method: "GET" });
      const nextTokenBalanceInfoJson = await nextTokenBalanceInfo.json();
      tokenBalances.push(...nextTokenBalanceInfoJson.tokens);
      nextLink = nextTokenBalanceInfoJson.links.next;
    }

    return tokenBalances;
  }

  // Purpose: get token info for a token
  // Returns: a MirrorNodeTokenInfo
  async getTokenInfo(tokenId: string) {
    const tokenInfo = await fetch(`${this.url}/api/v1/tokens/${tokenId}`, { method: "GET" });
    const tokenInfoJson = await tokenInfo.json() as MirrorNodeTokenInfo;
    return tokenInfoJson;
  }

  // Purpose: get NFT Infor for an account
  // Returns: an array of NFTInfo
  async getNftInfo(accountId: AccountId) {
    const nftInfo = await fetch(`${this.url}/api/v1/accounts/${accountId}/nfts?limit=100, { method: "GET" });
    const nftInfoJson = await nftInfo.json();

    const nftInfos = [...nftInfoJson.nfts] as MirrorNodeNftInfo[];

    // because the mirror node API paginates results, we need to check if there are more results
    // if links.next is not null, then there are more results and we need to fetch them until links.next is null
    let nextLink = nftInfoJson.links.next;
    while (nextLink !== null) {
      const nextNftInfo = await fetch(`${this.url}${nextLink}`, { method: "GET" });
      const nextNftInfoJson = await nextNftInfo.json();
      nftInfos.push(...nextNftInfoJson.nfts);
      nextLink = nextNftInfoJson.links.next;
    }
  }
}

```

```

    }
    return nftInfos;
}

// Purpose: get token balances for an account with token info in order to
display token balance, token type, decimals, etc.
// Returns: an array of MirrorNodeAccountTokenBalanceWithInfo
async getAccountTokenBalancesWithTokenInfo(accountId: AccountId):
Promise<MirrorNodeAccountTokenBalanceWithInfo[]> {
    //1. Retrieve all token balances in the account
    const tokens = await this.getAccountTokenBalances(accountId);
    //2. Create a map of token IDs to token info and fetch token info for each
token
    const tokenInfos = new Map<string, MirrorNodeTokenInfo>();
    for (const token of tokens) {
        const tokenInfo = await this.getTokenInfo(token.tokenid);
        tokenInfos.set(tokenInfo.tokenid, tokenInfo);
    }

    //3. Fetch all NFT info in account
    const nftInfos = await this.getNftInfo(accountId);

    //4. Create a map of token IDs to arrays of serial numbers
    const tokenIdToSerialNumbers = new Map<string, number[]>();
    for (const nftInfo of nftInfos) {
        const tokenId = nftInfo.tokenid;
        const serialNumber = nftInfo.serialnumber;

        // if we haven't seen this tokenId before, create a new array with the
serial number
        if (!tokenIdToSerialNumbers.has(tokenId)) {
            tokenIdToSerialNumbers.set(tokenId, [serialNumber]);
        } else {
            // if we have seen this tokenId before, add the serial number to the
array
            tokenIdToSerialNumbers.get(tokenId)!.push(serialNumber);
        }
    }

    //5. Combine token balances, token info, and NFT info and return
    return tokens.map(token => {
        return {
            ...token,
            info: tokenInfos.get(token.tokenid)!,
            nftSerialNumbers: tokenIdToSerialNumbers.get(token.tokenid)
        }
    });
}

async getAccountInfo(accountId: AccountId) {
    const accountInfo = await fetch(`${this.url}/api/v1/accounts/${accountId},
{ method: "GET" });
    const accountInfoJson = await accountInfo.json();
    return accountInfoJson;
}
}

```

</details>

2.5 Add Token Association Support

Before a user can receive a new token, they must associate with it. This

association helps protect users from receiving unwanted tokens.

Open `src/services/wallets/mirrorNodeClient.ts` and paste the function below the `getAccountTokenBalancesWithTokenInfo` function.

```
Typescript
// Purpose: check if an account is associated with a token
// Returns: true if the account is associated with the token, false otherwise
async isAssociated(accountId: AccountId, tokenId: string) {
  const accountTokenBalance = await this.getAccountTokenBalances(accountId);
  return accountTokenBalance.some(token => token.tokenid === tokenId);
}
```

3. Adding in the User Interface

In this step, we'll copy and paste the `home.tsx` file, which contains all the necessary code for adding UI components that enable token transfers and association with a token.

Open `src/pages/Home.tsx` and replace the existing code by pasting the below code:

<details>

<summary>Home.tx file</summary>

```
Typescript
import { Button, MenuItem, TextField, Typography } from "@mui/material";
import { Stack } from "@mui/system";
import { useWalletInterface } from "../services/wallets/useWalletInterface";
import SendIcon from '@mui/icons-material/Send';
import { useEffect, useState } from "react";
import { AccountId, TokenId } from "@hashgraph/sdk";
import { MirrorNodeAccountTokenBalanceWithInfo, MirrorNodeClient } from
"../services/wallets/mirrorNodeClient";
import { appConfig } from "../config";

const UNSELECTEDSERIALNUMBER = -1;

export default function Home() {
  const { walletInterface, accountId } = useWalletInterface();
  const [toAccountId, setToAccountId] = useState("");
  const [amount, setAmount] = useState<number>(0);
  // include all of this necessary for dropdown
  const [availableTokens, setAvailableTokens] =
useState<MirrorNodeAccountTokenBalanceWithInfo[]>([]);
  const [selectedTokenId, setSelectedTokenId] = useState<string>('');
  const [serialNumber, setSerialNumber] =
useState<number>(UNSELECTEDSERIALNUMBER);

  const [tokenIdToAssociate, setTokenIdToAssociate] = useState("");

  // include all of this necessary for dropdown
  // Purpose: Get the account token balances with token info for the current
account and set them to state
  useEffect(() => {
    if (accountId === null) {
      return;
    }
    const mirrorNodeClient = new MirrorNodeClient(appConfig.networks.testnet);
    // Get token balance with token info for the current account
    mirrorNodeClient.getAccountTokenBalancesWithTokenInfo(AccountId.fromString(accountId)).then((tokens) => {
```

```

    // set to state
    setAvailableTokens(tokens);
    console.log(tokens);
  }).catch((error) => {
    console.error(error);
  });
}, [accountId])

// include all of this necessary for dropdown
// Filter out tokens with a balance of 0
const tokensWithNonZeroBalance = availableTokens.filter((token) =>
token.balance > 0);
// include all of this necessary for dropdown
// Get the selected token balance with info
const selectedTokenBalanceWithInfo = availableTokens.find((token) =>
token.tokenid === selectedTokenId);

// include all of this necessary for dropdown
// reset amount and serial number when token id changes
useEffect(() => {
  setAmount(0);
  setSerialNumber(UNSELECTEDSERIALNUMBER);
}, [selectedTokenId]);

return (
  <Stack alignItems="center" spacing={4}>
    <Typography
      variant="h4"
      color="white"
    >
      Let's build a dApp on Hedera
    </Typography>
    {walletInterface !== null && (
      <>
        <Stack
          direction='row'
          gap={2}
          alignItems='center'
        >
          <Typography>
            Transfer
          </Typography>
          <TextField
            label='Available Tokens'
            value={selectedTokenId}
            select
            onChange={(e) => setSelectedTokenId(e.target.value)}
            sx={{
              width: '250px',
              height: '50px',
            }}
          >
            <MenuItem
              value={' '}
            >
              Select a token
            </MenuItem>
            {tokensWithNonZeroBalance.map((token) => {
              const tokenBalanceAdjustedForDecimals = token.balance /
Math.pow(10, Number.parseInt(token.info.decimals));
              return (
                <MenuItem
                  key={token.tokenid}
                  value={token.tokenid}

```

```

        >
        {token.info.name}({token.tokenid}):
({tokenBalanceAdjustedForDecimals})
    </MenuItem>
    );
  }
})
</TextField>
{selectedTokenBalanceWithInfo?.info?.type === "NONFUNGIBLEUNIQUE" &&
(
  <TextField
    label='Serial Number'
    select
    value={serialNumber.toString()}
    onChange={(e) =>
setSerialNumber(Number.parseInt(e.target.value))}
    sx={{
      width: '190px',
      height: '50px',
    }}
  >
    <MenuItem
      value={UNSELECTEDSERIALNUMBER}
    >
      Select a Serial Number
    </MenuItem>

{selectedTokenBalanceWithInfo.nftSerialNumbers?.map((serialNumber) => {
  return (
    <MenuItem
      key={serialNumber}
      value={serialNumber}
    >
      {serialNumber}
    </MenuItem>
  );
})}
  </TextField>
)}
{selectedTokenBalanceWithInfo?.info?.type === "FUNGIBLECOMMON" && (
  <TextField
    type='number'
    label='amount'
    value={amount}
    onChange={(e) => setAmount(parseInt(e.target.value))}
    sx={{
      maxWidth: '100px'
    }}
  />
)}
{/ not included in the dropdown stage. this is in the
association/send stage /}
<Typography>
  HTS Token
  to
</Typography>
<TextField
  value={toAccountId}
  onChange={(e) => setToAccountId(e.target.value)}
  label='account id or evm address'
/>
<Button
  variant='contained'

```

```

onClick={async () => {
  if (selectedTokenBalanceWithInfo === undefined) {
    console.log(Token Id is empty.)
    return;
  }

  // check if receiver has associated
  const mirrorNodeClient = new
MirrorNodeClient(appConfig.networks.testnet);
  const isAssociated = await
mirrorNodeClient.isAssociated(AccountId.fromString(toAccountId),
selectedTokenId);
  if (!isAssociated) {
    console.log(Receiver is not associated with token id: $
{selectedTokenId});
    return;
  }
  if (selectedTokenBalanceWithInfo.info.type ===
"NONFUNGIBLEUNIQUE") {
    await walletInterface.transferNonFungibleToken(
      AccountId.fromString(toAccountId),
      TokenId.fromString(selectedTokenId),
      serialNumber);
  } else {
    const amountWithDecimals = amount Math.pow(10,
Number.parseInt(selectedTokenBalanceWithInfo.info.decimals));
    await walletInterface.transferFungibleToken(
      AccountId.fromString(toAccountId),
      TokenId.fromString(selectedTokenId),
      amountWithDecimals);
  }
}}
>
  <SendIcon />
</Button>
</Stack>
<Stack
  direction='row'
  gap={2}
  alignItems='center'
>
  <TextField
    value={tokenIdToAssociate}
    label='token id'
    onChange={(e) => setTokenIdToAssociate(e.target.value)}
  />
  <Button
    variant='contained'
    onClick={async () => {
      if (tokenIdToAssociate === "") {
        console.log(Token Id is empty.)
        return;
      }
      await
walletInterface.associateToken(TokenId.fromString(tokenIdToAssociate));
    }}
  >
    Associate Token
  </Button>
</Stack>
</>
)}
</Stack>
)

```

```
}
```

```
</details>
```

```
{% hint style="info" %}
```

The crucial part of the code is found within the following code:

```
Typescript
// include all of this necessary for dropdown
// Purpose: Get the account token balances with token info for the current
account and set them to state
useEffect(() => {
  if (accountId === null) {
    return;
  }
  const mirrorNodeClient = new MirrorNodeClient(appConfig.networks.testnet);
  // Get token balance with token info for the current account

mirrorNodeClient.getAccountTokenBalancesWithTokenInfo(AccountId.fromString(accountId)).then((tokens) => {
  // set to state
  setAvailableTokens(tokens);
  console.log(tokens);
}).catch((error) => {
  console.error(error);
});
}, [accountId])
```

This code fetches and updates the list of tokens a user owns providing the available tokens to the dropdown menu.

```
{% endhint %}
```

4. Testing DApp Functionality

The application is ready to be started and tested. You will be testing:

- [] Connect to a DApp with a Hedera native wallet through WalletConnect
- [] Associate a token
- [] Transfer a token to a receiver account

4.1 Test Setup

You'll be creating four Hedera Testnet accounts, each with a balance of 10 HBAR. Two of these accounts will come pre-loaded with their own fungible tokens, and four accounts will come pre-loaded with their own non-fungible tokens (NFTs).

Open a new terminal window and create a new directory and change into that directory

```
shell
mkdir hedera-test-accounts && cd hedera-test-accounts
```

Open hedera-test-accounts folder in a new visual studio code window.

Create a new file and name it .env with the following contents. Remember to enter your account ID and your private key.

```
shell
MYACCOUNTID=<enter your account id>
MYPRIVATEKEY=<enter your DER private key>
```

Within the hedera-test-accounts home directory, execute the following command in the terminal,

```
shell
npx github:/hedera-dev/hedera-create-account-and-token-helper
```

```
{% hint style="danger" %}
Keep this terminal open for the remainder of the tutorial, as you will refer
back to it.
{% endhint %}
```

```
<details>
<summary>Test Accounts Sample Output </summary>
```

```
JSON
{
  "ed25519": {
    "sender": {
      "accountId": "0.0.xxxxxxxx",
      "privateKey": "302...",
      "FungibleTokenId": "0.0.xxxxxxxx",
      "NftTokenId": "0.0.xxxxxxxx"
    },
    "receiver": {
      "accountId": "0.0.xxxxxxxx",
      "privateKey": "302..."
    }
  },
  "ecdsaWithAlias": {
    "sender": {
      "accountId": "0.0.xxxxxxxx",
      "privateKey": "...hexadecimal string of length 64...",
      "FungibleTokenId": "0.0.xxxxxxxx",
      "NftTokenId": "0.0.xxxxxxxx"
    },
    "receiver": {
      "accountId": "0.0.xxxxxxxx",
      "privateKey": "...hexadecimal string of length 64..."
    }
  }
}
```

```
</details>
```

4.2 Import Sender and Receiver accounts

Import the sender and receiver accounts that were just outputted into your preferred wallet application. (MetaMask, HashPack, Blade, or Kabila)

For assistance on how to import a Hedera account into MetaMask refer to our [documentation](#) here.

```
{% hint style="danger" %}
Rename your imported accounts within your preferred wallet to keep track which
```


is the sender and receiver account.
{% endhint %}

4.3 Start the DApp

Navigate back to your application in Visual Studio Code, and in the terminal, run the following command

```
shell  
npm run start
```

<figure><figcaption><p>DApp homepage</p></figcaption></figure>

4.4 Connect to DApp as the Receiver

Click the Connect Wallet button in the upper right and select MetaMask and select the Sender account.

<figure><figcaption><p>Select your wallet</p></figcaption></figure>

<figure><figcaption><p>Connect with receiver account</p></figcaption></figure>

<figure><figcaption><p>Successfully connected to dapp view</p></figcaption></figure>

<details>
<summary>WalletConnect Instructions</summary>

Open HashPack and unlock your wallet

Go back to the DApp homepage and click 'Connect Wallet'

Choose WalletConnect

Copy the connection string

<figure><figcaption><p>Walletconnect modal</p></figcaption></figure>

Open HashPack and connect to DApp by clicking on the world in the upper right

<figure><figcaption><p>HashPack connect to dapp modal</p></figcaption></figure>

Paste the walletconnect string into HashPack pairing string textbox

<figure><figcaption><p>HashPack paste pairing string</p></figcaption></figure>

Select the Receiver account to connect with the DApp

<figure><figcaption><p>HashPack select account</p></figcaption></figure>

✔ You're connected!

</details>

4.5 Associate Receiver Account with Sender Account NFT

Open the output of the test accounts you created earlier and copy the `ecdsaWithAlias` Sender's account `NftTokenId`

Paste the `NftTokenId` in the DApps associate token textbox and click the button `Associate`

<figure><figcaption><p>Associate textbox</p></figcaption></figure>

MetaMask will prompt you to sign the transaction. If the extension does not automatically open, you will need to manually click on the MetaMask extension.

<figure><figcaption><p>Sign the Associate transaction</p></figcaption></figure>

Confirm the transaction

{% hint style="warning" %}
The react template uses the Hashio JSON RPC Relay URL to work with MetaMask. If you are experiencing degraded performance, follow this guide to switch to Arkhia or set up your own JSON RPC Relay. Edit the `src/config/networks.ts` with the new JSON RPC Relay URL.
{% endhint %}

4.6 Transfer NFT to Receiver Account

Disconnect as the Receiver account and reconnect with the Sender account. To do this, open the MetaMask extension, click on the three dots in the upper right, select "Connected Sites," and then disconnect the Receiver account. All other wallets disconnect by clicking on your account ID in the upper right of the DApp homepage.

Connect to the DApp as the Sender Account.

As the Sender,

- Select from available tokens the HederaNFT

- Select the NFT with serial number 5 from the drop-down menu.

- Enter the account ID or EVM address of the Receiver account.

- Click the "send" button.

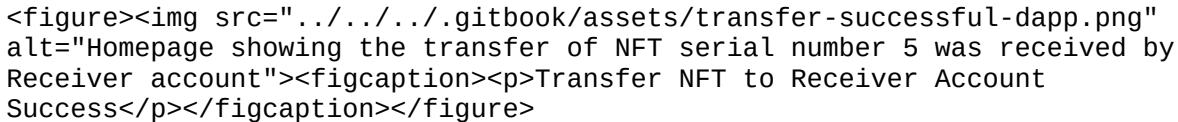
Sign the transaction on MetaMask to complete the transfer of the NFT from the Sender to the receiver account.

<figure><figcaption><p>Transfer NFT to Receiver Account</p></figcaption></figure>

4.7 Verify Receiver Account Received the NFT

Disconnect as the Sender account and reconnect as the Receiver account.

Check the dropdown menu and ensure the Receiver account has NFT serial number 5.


Transfer NFT to Receiver Account
Success

Try with HashPack, Blade or Kabila

Optionally, import your accounts into any of the above Hedera native wallets and test out transferring more tokens.

Complete

🎉 Congratulations! You have successfully walked through creating a Hedera DApp that transfers HTS tokens using MetaMask, HashPack, Blade, or Kabila.

You have learned how to:

- [x] Query the mirror node for account token balance, token information, and Non-fungible information.

- [x] Query the mirror node to check if the receiver has associated with a token ID

- [x] Associate an HTS token with HashPack, Kabila, Blade, or MetaMask through a UI

- [x] Transfer an HTS token through a UI

<p>Writer: Abi Castro, DevRel Engineer</p> <p>GitHub LinkedIn</p>

how-to-auto-create-hedera-accounts-with-hbar-and-token-transfers.md:

How to Auto-Create Hedera Accounts with HBAR and Token Transfers

HIP-32 introduced the ability to auto-create accounts when sending HBAR to an alias that does not exist on the network. When HBAR is sent to an alias that does not exist on the network, the account creation fee is deducted from the HBAR sent and the account is auto-created. The new account's initial balance is (sent HBAR - account creation fee). This new method of account creation allowed wallet providers to create free "accounts" to users. However, if a user sends fungible tokens or NFT's to an alias, it would result in an INVALID\ACCOUNT\ID error because the alias does not exist on the network. The auto-account creation flow could not deduct the account creation fee from an HTS token; the account creation fee must be paid in HBAR.

HIP-542 provides a solution to allow sending HTS tokens to an alias that does not exist on the network. This is achieved by charging the account creation fee to the transfer transaction payer. In addition, there will be one auto-association slot included in the transaction for the new account to associate with the HTS token. You won't have to first create the account, complete a token association, and then finally do a token transfer.

Furthermore, this change also applies to sending HBAR to an alias. Instead of deducting the creation fee from the sent HBAR, it will be deducted from the payer of the transfer transaction. The new account balance will receive the sent HBAR amount in full. Learn more here.

The figure below highlights the transaction flow before HIP-542 and after.

```
<figure><figcaption></figcaption></figure>
```

In this tutorial, a treasury account will be created to transfer HTS tokens to Bob's ECDSA public key alias, involving a transfer of 10 FT and 1 NFT. The tutorial will also cover creating fungible tokens and an NFT collection (1000 FT / 5 NFT), creating Bob's ECDSA public key alias, and returning Bob's new account ID while confirming their ownership of the transferred tokens and NFT.

Prerequisites

Get a Hedera Testnet account here.
Set up your environment and create a client here.

<details>

<summary>Get the example code on GitHub:</summary>

auto-create account by sending FT
auto-create account by sending NFT

</details>

Table of Contents

1. Create Treasury Account
2. Create FTs and NFT Collection
3. Create Bob's ECDSA Public Key Alias
4. Return New Account ID

Create Treasury Account

We create the treasury account, which will be the holder of the fungible and non-fungible tokens. The treasury account will be created with an initial balance of 5 HBAR.

```
javascript  
const [treasuryAccId, treasuryAccPvKey] = await createAccount(client, 5);
```

Set Up Helper Functions

We will create the functions necessary to create a new account, create fungible tokens, and create a new NFT collection. Use the tabs to see the helper functions.

```
{% tabs %}  
{% tab title="Create Account With Initial Balance" %}  
javascript  
const createAccount = async (client: Client, initialBalance: number) => {  
  const accountPrivateKey = PrivateKey.generateED25519();  
  
  const response = await new AccountCreateTransaction()
```

```

        .setInitialBalance(new Hbar(initialBalance))
        .setKey(accountPrivateKey)
        .execute(client);

const receipt = await response.getReceipt(client);

return [receipt.accountId, accountPrivateKey];
};

{% endtab %}

{% tab title="Create Simple Fungible Token" %}
javascript
export const createFungibleToken = async (
  client: Client,
  treasuryAccId: string | AccountId,
  supplyKey: PrivateKey,
  treasuryAccPvKey: PrivateKey,
  initialSupply: number,
  tokenName: string,
  tokenSymbol: string,
): Promise<TokenId> => {
  /
    Create a transaction with token type fungible
    Returns Fungible Token Id
  /
  const createTokenTxn = await new TokenCreateTransaction()
    .setTokenName(tokenName)
    .setTokenSymbol(tokenSymbol)
    .setTokenType(TokenType.FungibleCommon)
    .setInitialSupply(initialSupply)
    .setTreasuryAccountId(treasuryAccId)
    .setSupplyKey(supplyKey)
    .setMaxTransactionFee(new Hbar(30))
    .freezeWith(client); //freeze tx from from any further mods.

  const createTokenTxnSigned = await createTokenTxn.sign(treasuryAccPvKey);
  // submit txn to heder network
  const txnResponse = await createTokenTxnSigned.execute(client);
  // request receipt of txn
  const txnRx = await txnResponse.getReceipt(client);
  const txnStatus = txnRx.status.toString();
  const tokenId = txnRx.tokenId;
  if (tokenId === null) {
    throw new Error("Somehow tokenId is null");
  }

  console.log(
    Token Type Creation was a ${txnStatus} and was created with token id: $
    {tokenId}
  );

  return tokenId;
};

{% endtab %}

{% tab title="Create Simple Non-Fungible Token" %}
javascript
export const createNonFungibleToken = async (
  client: Client,
  treasuryAccId: string | AccountId,
  supplyKey: PrivateKey,
  treasuryAccPvKey: PrivateKey,

```

```

initialSupply: number,
tokenName: string,
tokenSymbol: string,
): Promise<[TokenId | null, string]> => {
/
    Create a transaction with token type fungible
    Returns Fungible Token Id and Token Id in solidity format
/
const createTokenTxn = await new TokenCreateTransaction()
    .setTokenName(tokenName)
    .setTokenSymbol(tokenSymbol)
    .setTokenType(TokenType.NonFungibleUnique)
    .setDecimals(0)
    .setInitialSupply(initialSupply)
    .setTreasuryAccountId(treasuryAccId)
    .setSupplyKey(supplyKey)
    .setAdminKey(treasuryAccPvKey)
    .setMaxTransactionFee(new Hbar(30))
    .freezeWith(client); //freeze tx from from any further mods.

const createTokenTxnSigned = await createTokenTxn.sign(treasuryAccPvKey);
// submit txn to hedera network
const txnResponse = await createTokenTxnSigned.execute(client);
// request receipt of txn
const txnRx = await txnResponse.getReceipt(client);
const txnStatus = txnRx.status.toString();
const tokenId = txnRx.tokenId;
if (tokenId === null) { throw new Error("Somehow tokenId is null."); }

const tokenIdInSolidityFormat = tokenId.toSolidityAddress();

console.log(
    Token Type Creation was a ${txnStatus} and was created with token id: $
{tokenId}
);

return [tokenId, tokenIdInSolidityFormat];
};

{% endtab %}

{% tab title="Mint Token And Create NFT Collection" %}
javascript
const mintTokenTxn = new TokenMintTransaction()
    .setTokenId(tokenId)
    .setMetadata(metadata)
    .freezeWith(client);

const mintTokenTxnSigned = await mintTokenTxn.sign(supplyKey);

// submit txn to hedera network
const txnResponse = await mintTokenTxnSigned.execute(client);

const mintTokenRx = await txnResponse.getReceipt(client);
const mintTokenStatus = mintTokenRx.status.toString();

console.log(Token mint was a ${mintTokenStatus});
};

export const createNewNftCollection = async (
    client: Client,
    tokenName: string,
    tokenSymbol: string,

```

```

metadataIPFSUrls: Buffer[], // already uploaded ipfs metadata json files
treasuryAccountId: string | AccountId,
treasuryAccountPrivateKey: PrivateKey,
): Promise<{
  tokenId: TokenId,
  supplyKey: PrivateKey,
}> => {
  // generate supply key
  const supplyKey = PrivateKey.generateECDSA();

  const [tokenId,] = await createNonFungibleToken(client, treasuryAccountId,
supplyKey, treasuryAccountPrivateKey, 0, tokenName, tokenSymbol);
  if (tokenId === null || tokenId === undefined) {
    throw new Error("Somehow tokenId is null");
  }

  const metadatas: Uint8Array[] = metadataIPFSUrls.map(url => Buffer.from(url));

  // mint token
  await mintToken(client, tokenId, metadatas, supplyKey);
  return {
    tokenId: tokenId,
    supplyKey: supplyKey,
  };
}

{% endtab %}
{% endtabs %}

```

Create FTs and an NFT Collection

Leverage the createFungibleToken helper function defined above to create 10000 "Hip-542 example" fungible tokens. Use the code tab switch on the upper left of the code block to see how we use createNewNftCollection to create our new NFT collection consisting of 5 NFTs.

```

{% tabs %}
{% tab title="CreateFungibleToken" %}
javascript
const tokenId = await createFungibleToken(client, treasuryAccId, supplyKey,
treasuryAccPvKey, 10000, 'HIP-542 Token', 'H542');

{% endtab %}

{% tab title="CreateNewNftCollection" %}
javascript
// IPFS content identifiers for the NFT metadata
const metadataIPFSUrls: Buffer[] = [

Buffer.from("ipfs://bafkreiap62fsqxmo4hy45bmwiqolqqtkhtehghqauixvv5mcq7uofdpvt4"
),

Buffer.from("ipfs://bafkreibvlvlf36lilrqaum54ga3nlumms34m4kab2x67f5piofmo5fsa"
),

Buffer.from("ipfs://bafkreidrqr67amvygjnvr2mgdggg2alaowoy34ljubot6qwf6bcf4yma4"
),

Buffer.from("ipfs://bafkreicoorrcx3d4foreggz72aedxhosuk3cjgumglstokuhw2cmz22n7u"
),

Buffer.from("ipfs://bafkreidv7k5vfn6gnj5mhahnrvhxep4okw75dwbt6o4r3rhe3ktraddf5a"

```

```

),
];

/
  Step 2
  Create nft collection
/
const nftCreateTxnResponse = await createNewNftCollection(client, 'HIP-542
Example Collection', 'HIP-542', metadataIPFSUrls, treasuryAccId,
treasuryAccPvKey);

{% endtab %}
{% endtabs %}

```

Create Bob's ECDSA Public Key Alias

An alias is an initial public key that will convert into a Hedera account through auto-account creation. An alias consists of `\<shard>.\<realm>.\<bytes>.`

To learn more about accounts created via an account alias go [here](#).

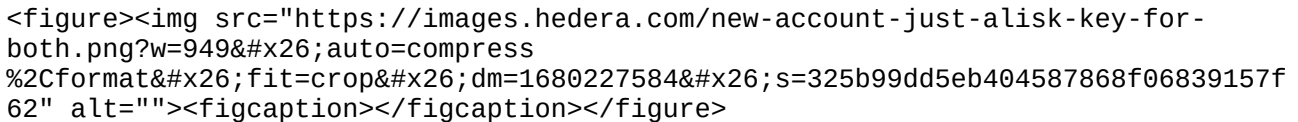
```

javascript
const privateKey = PrivateKey.generateECDSA();
const publicKey = privateKey.publicKey;

// Assuming that the target shard and realm are known.
// For now they are virtually always 0 and 0.
const aliasAccountId = publicKey.toAccountId(0, 0);

console.log(- New account ID: ${aliasAccountId.toString()});
if (aliasAccountId.aliasKey === null) { throw new Error('alias key is empty') }
console.log(- Just the aliasKey: ${aliasAccountId.aliasKey.toString()}\n);

```

A screenshot of a web browser displaying a Hedera account creation page. The page shows a 'New account ID' and an 'alias key'. The account ID is '0.0.1680227584.325b99dd5eb404587868f06839157f62'. The alias key is '0.0.1680227584.325b99dd5eb404587868f06839157f62'. The page title is 'New account - just an alias key for both'.

Set up helper functions for transferring HTS tokens

Once we have our treasury account with FT and a new NFT collection created, our next step is to transfer them to Bob using their alias. We'll create the `sendToken` helper function to send fungible tokens and `createTransferNft` to send a single NFT.

A quick reminder to use the tab on the left of the code block to switch between the two helper functions.

```

{% tabs %}
{% tab title="SendToken" %}
javascript
export const sendToken = async (client: Client, tokenId: TokenId, owner:
AccountId, aliasAccountId: AccountId, sendBalance: number, treasuryAccPvKey:
PrivateKey) => {
  const tokenTransferTx = new TransferTransaction()
    .addTokenTransfer(tokenId, owner, -sendBalance)
    .addTokenTransfer(tokenId, aliasAccountId, sendBalance)
    .freezeWith(client);

  // Sign the transaction with the operator key
  let tokenTransferTxSign = await tokenTransferTx.sign(treasuryAccPvKey);

```



```

    // Submit the transaction to the Hedera network
    let tokenTransferSubmit = await tokenTransferTxSign.execute(client);

    // Get transaction receipt information
    await tokenTransferSubmit.getReceipt(client);
}

{% endtab %}

{% tab title="TransferNft" %}
javascript
export const transferNft = async (client: Client, nftTokenId: TokenId, nftId:
number, treasuryAccId: AccountId, treasuryAccPvKey: PrivateKey, aliasAccountId:
AccountId) => {
  const nftTransferTx = new TransferTransaction()
    .addNftTransfer(nftTokenId, nftId, treasuryAccId, aliasAccountId)
    .freezeWith(client);

  // Sign the transaction with the treasury account private key
  const nftTransferTxSign = await nftTransferTx.sign(treasuryAccPvKey);

  // Submit the transaction to the Hedera network
  const nftTransferSubmit = await nftTransferTxSign.execute(client);

  // Get transaction receipt information here
  await nftTransferSubmit.getReceipt(client);
}

{% endtab %}
{% endtabs %}

```

Transfer FT and an NFT to Bob using their alias

Transfer 5 fungible tokens to Bob using their alias and the helper function
sendToken.

Transfer the NFT with serial number 1 to Bob using the helper function
transferNFT.

```

{% tabs %}
{% tab title="Send Bob 10 FT" %}
javascript
await sendToken(client, tokenId, treasuryAccId, aliasAccountId, 5,
treasuryAccPvKey);

{% endtab %}

{% tab title="Send Bob An NFT" %}
javascript
const nftTokenId = nftCreateTxnResponse.tokenId;
const exampleNftId = 1;
await transferNft(client, nftTokenId, exampleNftId, treasuryAccId,
treasuryAccPvKey, aliasAccountId);

{% endtab %}
{% endtabs %}

```

Return New Account ID

Create a helper function to return the corresponding account Id to the given an
alias.

```

javascript
export const getAccountIdByAlias = async (client: Client, aliasAccountId:
AccountId ) => {
  const accountInfo = await new AccountInfoQuery()
    .setAccountId(aliasAccountId)
    .execute(client);

  return accountInfo.accountId;
}

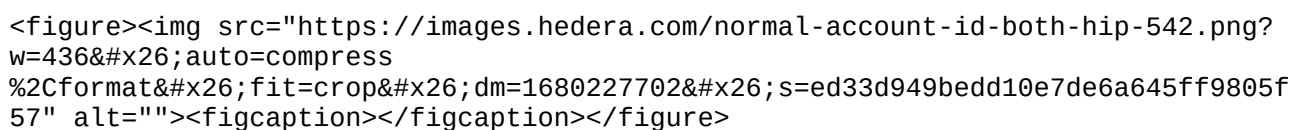
```

Next we call `getAccountIdByAlias` and pass in our client and Bob's alias as the arguments.

```

javascript
const accountId = await getAccountIdByAlias(client, aliasAccountId);
console.log(The normal account ID of the given alias: ${accountId});

```



Show Bob's new account owns the 5 FT tokens

Complete an `AccountBalanceQuery` to show that Bob's new account owns the 5 fungible tokens the treasury account sent.

```

javascript
const accountBalances = await new AccountBalanceQuery()
  .setAccountId(aliasAccountId)
  .execute(client);

if (!accountBalances.tokens || !accountBalances.tokens.map) {
  throw new Error('account balance shows no tokens.')
}

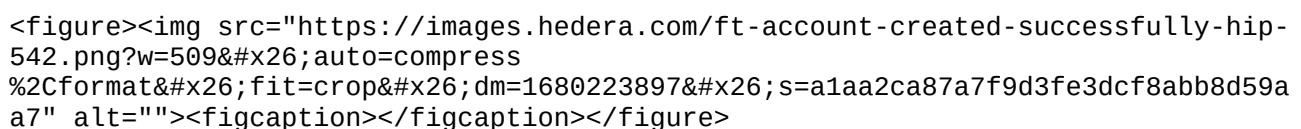
const tokenBalanceAccountId = accountBalances.tokens.map
  .get(tokenId.toString());

if (!tokenBalanceAccountId) {
  throw new Error(account balance does not have tokens for token id: $
{tokenId}.);
}

tokenBalanceAccountId.toInt() === 5
? console.log(
  Account is created successfully using HTS 'TransferTransaction'
)
: console.log(
  "Creating account with HTS using public key alias failed"
);

client.close();

```



Show Bob's new account owns the NFT

First, create a helper function that creates a TokenNftInfoQuery transaction and returns the account id of the NFT owner for a specific NFT serial number.

```
javascript
export const getNftOwnerByNftId = async (client: Client, nftTokenId: TokenId,
exampleNftId: number) => {
  const nftInfo = await new TokenNftInfoQuery()
    .setNftId(new NftId(nftTokenId, exampleNftId))
    .execute(client);

  if (nftInfo === null) { throw new Error('nftInfo is null.') }
  const nftOwnerAccountId = nftInfo[0].accountId.toString();
  console.log(- Current owner account id: ${nftOwnerAccountId} for NFT with
serial number: ${exampleNftId});
  return nftOwnerAccountId;
}
```


Then call getNftOwnerByNft and do a simple check to ensure the account id returned matches the account id created when we sent the NFT to Bob's alias.

```
javascript
const nftOwnerAccountId = await getNftOwnerByNftId(client, nftTokenId,
exampleNftId);

nftOwnerAccountId === accountId
  ? console.log(
    The NFT owner accountId matches the accountId created with the HTS\n
  )
  : console.log(The two account IDs does not match\n);

client.close();
```

<figure><figcaption></figcaption></figure>

Console Output 

<figure><figcaption></figcaption></figure>

And that's a wrap! 📦 You've completed sending HTS tokens to an alias and triggering an auto-account creation! As well as learned that the account creation fee is paid by the payer of the transfer transaction.

Join and collaborate with Hedera Developers on the Hedera Discord Server! \

\

Happy Building! 🧡

<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Abi, DevRel Engineer</p><p>GitHub | LinkedIn</p></td><td align="center"><p>https://www.linkedin.com/in/a-ridley/</td></tr><tr><td align="center"><p>Editor: Krystal, Technical Writer</p><p>GitHub | <a

```
href="https://hashnode.com/@theekrystallee">Hashnode</a></p></td><td><a  
href="https://twitter.com/theekrystallee">https://twitter.com/theekrystallee</  
a></td></tr></tbody></table>
```

how-to-configure-a-mirror-node-and-query-specific-data.md:

```
---  
description: >-  
  A step-by-step guide on how to configure a mirror node and query specific data  
  on the Hedera Network  
---
```

How to Configure a Mirror Node and Query Data

Hedera Mirror Node is a useful tool that lets developers and users access past transaction data on the Hedera network. You can view and analyze network data such as transactions, transfers, balances, and events from the past in a reliable, scalable, and efficient way. The best feature is the capability to configure your mirror node to query only data that meet your tailored specifications.

This guide provides step-by-step instructions on how to configure and use a Hedera Mirror Node to access past transaction data on the Hedera network. You will learn how to configure your mirror node to store only the latest 90 days of data or data for a specific entity (account, smart contract, etc.) and how to use basic SQL queries to analyze the data.

Prerequisites

Basic understanding of Hedera Mirror Nodes.
Basic understanding of terminal commands and SQL.
Java (openjdk@17: Java version 17), Gradle (the latest version), and PostgreSQL (the latest version) are installed on your machine.
Docker (>= v20.10.x) installed and open on your machine. Run `docker -v` in your terminal to check the version you have installed.

Table of Contents

1. Set Up Mirror Node
2. Configure Mirror Node
3. Start Mirror Node
4. Query Mirror Node
5. Additional Resources

Set Up Mirror Node

Clone the Hedera mirror node repository and navigate to the project directory:

```
bash  
git clone https://github.com/hashgraph/hedera-mirror-node.git  
cd hedera-mirror-node
```

{% hint style="info" %}

Note: Cloning the mirror node repository could require some time to complete.

{% endhint %}

Configure Mirror Node

In this example, we will configure the mirror node to store the last 90 days of data or data for a specific account ID (entity). To achieve this, we will need to create and modify an `application.yml` file. This file contains configuration settings for the mirror node, such as which data to store and how long to store it.

First, create a new configuration folder and file inside the `hedera-mirror-importer` directory. The Mirror Node importer directory contains the source code for the importer tool, which allows users to import data from the Hedera Mainnet, Testnet, or Previewnet. This creates and imports a read-only instance of the Hedera network data stored in its own database.

Run the following command to create the right folder and file:

```
bash
mkdir hedera-mirror-importer/config
touch hedera-mirror-importer/config/application.yml
```

Transaction and entity filtering

The mirror node may be configured only to store a subset of data for entities and/or transaction types of interest – essentially, which rows of data to retain.

In this example, we'll use the `application.yml` format for demonstration purposes. This configuration retains transaction and crypto transfer data for 90 days, excludes data for entity `0.0.111478`, and includes specific transactions for entities `0.0.111710` and `0.0.111734`. Furthermore, it prevents the storage of topic data. You can check out the other two alternative formats here if you don't like working with the YAML format.

Breaking down application.yml configuration

Here's an overview the `application.yml` file. Copy the following lines into `application.yml` and save it.

```
{% code title="application.yml" %}
yaml
hedera:
  mirror:
    importer:
      network: DEMO
      retention:
        period: 90D
        frequency: 60S
        enabled: true
      include:
        - transaction
        - cryptotransfer
    parser:
      exclude:
        - entity: [0.0.111478]
      include:
        - transaction: [CRYPTOTRANSFER, CRYPTOCREATEACCOUNT]
        - entity: [0.0.111710]
          transaction: [CONTRACTCREATEINSTANCE]
        - entity: [0.0.111734]
    record:
      entity:
```

```
persist:
  topics: false
```

```
{% endcode %}
```

Here's a breakdown of what each section of the configuration file does:

behaviorhedera: This is the root section of the configuration file, indicating that the settings apply to the Hedera network.

mirror: This is a sub-section that pertains specifically to the Mirror node.

importer: This sub-section defines settings for the Mirror node's importer, which is responsible for retrieving transaction data from the network and storing it in a local database for querying.

importer.network: DEMO: This specifies that the importer should connect to a bucket with demo data. It's the easiest way to experiment with the mirror node and importer. If you want to connect to the TESTNET, MAINNET, or PREVIEWNET, you need to follow this tutorial.

importer.retention: This sub-section specifies the retention period and frequency for importing data. In this case, the importer will clean data that is older than 90 days every 60 seconds. If you omit the frequency key, the default behavior for cleaning data is once a day.

importer.retention.include: This specifies the database tables that should be included in the imported data. The tables specified are transaction and cryptotransfer. You can find all tables in the GitHub repository for the mirror node.

parser: This sub-section defines settings for the data parser, which determines the data that gets stored in the database or the data that should be filtered.

parser.exclude: This specifies the entities or transaction types that should be excluded from the imported data. In this case, the parser.exclude.entity with ID 0.0.111478 is excluded.

parser.include: This specifies the entities or transaction types that should be included from the imported data. In this case, the parser.include.entity with ID 0.0.111478 is included, and two specific transaction types (CRYPTOTRANSFER and CRYPTOCREATEACCOUNT) are included via parser.include.transaction.

You can also combine entity and transaction fields. In our example, we only want to store CONTRACTCREATEINSTANCE transactions for the entity with ID 0.0.111710.

parser.record: This sub-section specifies how the imported data should be recorded. In this case, the entity object is specified, which means that data should be recorded for each unique entity (account) involved in the transactions. The persist setting is set to false, which means that topic data for entities should not be persisted.

period: 90D: This indicates that the importer should retain the imported data for a period of 90 days. After this period, the data will be deleted.

```
{% hint style="info" %}
```

Note: The parser.exclude properties take priority over the parser.include properties. If you list the same value in both lists, it will be excluded.

```
\
```

In addition, the various boolean hedera.mirror.importer.record.entity.persist properties may be specified to control which additional fields get stored (which additional tables get recorded).

See the hedera.mirror.importer.parser.include. and hedera.mirror.importer.parser.exclude. properties listed in this table.

```
{% endhint %}
```

```
<details>
```

```
<summary>Alternative configuration formats</summary>
```

```
application.properties
```

To configure the above scenario via application.properties file, include the

following lines:

```
properties
hedera.mirror.importer.network=DEMO
hedera.mirror.importer.retention.period=90D
hedera.mirror.importer.retention.frequency=60S
hedera.mirror.importer.retention.enabled=true
hedera.mirror.importer.retention.include=transaction, cryptotransfer
hedera.mirror.importer.parser.exclude.entity=0.0.111478
hedera.mirror.importer.parser.include.transaction=CRYPTOTRANSFER, CRYPTOCREATEACCOUNT
hedera.mirror.importer.parser.include.entity.0.0.111710.transaction=CONTRACTCREATEINSTANCE
hedera.mirror.importer.parser.include.entity=0.0.111734
hedera.mirror.importer.record.entity.persist.topics=false
```

environment variables

To configure the above scenario via environmental variables, set the following:

```
REDERAMIRRORIMPORTERNETWORK=DEMO
REDERAMIRRORIMPORTERRETENTIONPERIOD=90D
REDERAMIRRORIMPORTERRETENTIONFREQUENCY=60S
REDERAMIRRORIMPORTERRETENTIONENABLED=true
REDERAMIRRORIMPORTERRETENTIONINCLUDE=transaction, cryptotransfer
REDERAMIRRORIMPORTERPARSEREXCLUDEENTITY=0.0.111478
REDERAMIRRORIMPORTERPARSERINCLUDETRANSACTION=CRYPTOTRANSFER, CRYPTOCREATEACCOUNT
REDERAMIRRORIMPORTERPARSERINCLUDEENTITY00111710TRANSACTION=CONTRACTCREATEINSTANCE
REDERAMIRRORIMPORTERPARSERINCLUDEENTITY=00111734
REDERAMIRRORIMPORTERRECORDENTITYPERSISTTOPICS=false
```

</details>

More details about retention [here](#) and transaction and entity filtering [here](#).

Start Mirror Node

The PostgreSQL container is responsible for creating the database for the mirror node instance, and the REST API container allows you to use the REST APIs to query the mirror node instance. The database stores the transaction data retrieved by the importer component of the mirror node, and the REST API provides an interface for accessing that data using HTTP requests. The importer component is responsible for retrieving the transaction data from the Hedera network and storing it in the database. Let's start up the database!

1. Start the database

Open Docker and start the PostgreSQL and REST API containers in the root directory:

```
bash
docker compose up -d db rest && docker logs hedera-mirror-node-db-1 --follow
```

Wait until you see the "database system is ready to accept connections" message in the console log, then control + c to terminate the current process.

<figure>

alt=""><figcaption><p>Console output after starting both containers.</p></figcaption></figure>

2. Run the importer

Now the database is ready, let's import demo data. Run the importer in the same root directory:

```
bash
./gradlew :importer:bootrun
```

This process may take some time, but once you see this in your console and the process is at 92%, you kill the process with control + c. If you let the process run, it will import more data that you don't need for this tutorial.

<figure><figcaption><p>Mirror Importer process console</p></figcaption></figure>

{% hint style="warning" %}

Note: Should you encounter an error during this step or if the command doesn't execute successfully, it's recommended to run the command again.

{% endhint %}

3. Connect to the PostgreSQL database

To connect the PostgreSQL database, we need to retrieve the database credentials. Open a new terminal window and run the following command in the root directory. Copy the database password that your console returns. You need this for the next step.

```
bash
cat docker-compose.yml | grep OWNER
```

In the same directory, run the following command that will connect the database:

```
bash
psql -U mirrornode -h 127.0.0.1
```

Enter the database password when prompted. If you successfully connect to the PostgreSQL database, your console should be ready to execute queries and look something like this:

```
hedera-mirror-node % psql -U mirrornode -h 127.0.0.1
Password for user mirrornode:
psql (14.6 (Homebrew), server 14.7)
Type "help" for help.
```

```
mirrornode=> <ENTER YOUR SQL QUERIES GO HERE>
```

Query Mirror Node

In this section, you can try out multiple queries that show you how to retrieve data from the PostgreSQL database. You need a basic understanding of SQL queries to craft your own queries.

{% hint style="info" %}

Most queries include the field type which refers to a transaction type, e.g. 11

refers to CRYPTOCREATEACCOUNT and 14 refers to CRYPTOTRANSFER. The most common transaction types are:

Type 7: CONTRACTCALL
Type 11: CRYPTOCREATEACCOUNT
Type 14: CRYPTOTRANSFER
Type 24: CONSENSUSCREATETOPIC
Type 27: CONSENSUSSUBMITMESSAGE
Type 29: TOKENCREATION
Type 37: TOKENMINT
Type 40: TOKENASSOCIATE

Check out the complete list of transaction types in the TransactionTypes.java file.

```
{% endhint %}
```

Execute the following queries to analyze the data stored in your local database.

```
sql
```

```
-- 1. query counts transactions based on their type and filters out
-- types 11 (CRYPTOCREATEACCOUNT) and 14 (CRYPTOTRANSFER)
SELECT count(), payeraccountid, entityid, type
FROM transaction WHERE type not in (11, 14)
GROUP by payeraccountid, entityid, type order by type;

-- 2. query counts transactions based on their payer account ID or entity ID
-- and filters for the value 111478
SELECT count(), payeraccountid, entityid, type
FROM transaction WHERE payeraccountid=111478 OR entityid=111478
GROUP by payeraccountid, entityid, type ORDER by type;

-- 3. query only type 8 -> CONTRACTCREATEINSTANCE (CRYPTOCREATEACCOUNT,
CRYPTOTRANSFER)
SELECT FROM transaction WHERE type=8;

-- 4. query count for each transaction type
SELECT count(), type
FROM transaction
GROUP by type;

-- 5. query shows different fields for the transaction table
SELECT payeraccountid, nodeaccountid, result, type, chargedtxfee,
transactionhash
FROM transaction
WHERE type not in (11, 14);
```

<details>

<summary>Query output verification  </summary>

Note that your output might differ depending on how much data you have imported into your database with the `./gradlew :importer:bootrun` command.

```
<pre><code>-- Query 1:
count | payeraccountid | entityid | type
-----+-----+-----+-----
    5 |           1117 |    61457 |    7
   71 |          111699 |   111704 |    7
    2 |          111721 |   111728 |    7
    2 |          111749 |   111757 |    7
    2 |          111478 |   111482 |    7
    1 |          111749 |   111757 |    8
(...)
```

-- Query 2:

count	payeraccountid	entityid	type
2	111478	111482	7
1	111478	111482	8
1	111478	111482	9
1	111478	111595	11
1	111478	111647	11
1	111478	111644	11

(...)

-- Query 3:

consensustimestamp	type	result	payeraccountid	validstartns	validdurationseconds	nodeaccountid	entityid	initialbalance	maxfee	chargedtxfee	memo	transactionhash	transactionbytes	scheduled	nonce	parentconsensustimestamp	errata	index												
1570800945787190002	8	22	111152	1570800933439239200	30	3	111157	0	5000000000	3581540281	\x637265617465436f6e7472616374	\xd226f8dd7922f9024331c5bf00da8e5bae6ade0db24fa4283e925ef9f06b4b4cddb1ff6511e552fe1a669a9b9cbaca8	f	0	1570801009263126000	8	22	111158	1570800996904028800	30	3	111164	0	9000000000	1991432867	\x637265617465436f6e7472616374	\xcb04329381668a481a0734cabfeac28b56f74cd9e3db42af031a47a6c98a598555d6d502623d237c679e00ee9b61dfc2	f	0	14

(...)

-- Query 4:

count	type
30	12
6676	14
34	17
13	8
572	11
10	9
82	7
146	19
7	18
4	16
894	15

(...)

-- Query 5:

payeraccountid	nodeaccountid	result	type	chargedtxfee	transactionhash
111146	3	22	17	608622391	\x9a8f13e9989462a2028bc5e3946d0f3d556d4ccfc65f2afa55b94c535823ea8a27caf630de85ae22303a73b7a0ded98d
111146	3	22	17	653540869	\

```

xd4964bfe66d311c60717c733e39c1e4b4aae38138e0f583fef9ae3844eb95772862f5f467ebb96e
fd06e80ba0b0059da
      111146 |           3 |      22 |   15 |           1256006 | \
x1c2b27dbf2d093fa353fc444e0a3ed1ac932518a37d2b03509f8a06039d454df82aeba5f35d7a1f
ea5babaca298a4288
(...)
</code></pre>

```

</details>

```

{% hint style="danger" %}
The PostgreSQL database is not an end product; the Hedera Mirror Node REST API
is. Hence, SQL queries might fail when the engineering team updates the
underlying database model in new releases. This tutorial has been tested with
release v0.79.1.
{% endhint %}

```

To exit the psql console, run the quit command:

```

bash
\q

```

Lastly, run the following command to stop and remove the created containers:

```

bash
docker compose down

```

Congratulations! 🎉 You have successfully learned how to configure the Hedera Mirror Node to query specific data. Feel free to reach out on Discord if you have any questions!

Additional Resources

🔗 Mirror Node Repository

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th><th data-hidden data-card-cover data-
type="files"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Krystal,
Technical Writer</p><p><a href="https://github.com/theekrystallee">GitHub</a> |
<a href="https://hashnode.com/@theekrystallee">Hashnode</a></p></td><td><a
href="https://twitter.com/theekrystallee">https://twitter.com/theekrystallee</
a></td><td></td></tr><tr><td align="center"><p>Editor: Michiel, Developer
Advocate</p><p><a href="https://github.com/michielmulders">GitHub</a> | <a
href="https://www.linkedin.com/in/michielmulders/">LinkedIn</a></p></td><td><a
href="https://www.linkedin.com/in/michielmulders/">https://www.linkedin.com/in/
michielmulders/</a></td><td></td></tr></tbody></table>

```

how-to-connect-metamask-to-hedera.md:

How to Connect MetaMask to Hedera

Download the MetaMask wallet, then configure the Hedera network/testnet settings in the MM network settings with one of the three methods below:

🔗 Connecting via HashScan


🔗 Manual Method

ChainList Method

Each method offers its own advantages and caters to different user preferences. By the end of this guide, you'll be able to connect your MetaMask wallet to the Hedera network.

1. Connecting via HashScan

This new method allows you to connect to the Hedera network through HashScan:

1. Go to <https://hashscan.io>.
2. Click on the Connect Wallet button in the top right corner.
3. Select  MetaMask from the list of wallet options.
4. If you haven't added the Hedera network to MetaMask yet, HashScan will prompt you to add it.
5. Click Approve to add the Hedera network to MetaMask.
6. Once added, you can switch to the Hedera network in MetaMask.

<div>

<figure><figcaption></figcaption></figure>

<figure><figcaption></figcaption></figure>

</div>

2. Manual Method

1. Open MetaMask and click on the network dropdown at the top.
2. Select Add Network and then Add Network Manually.
3. Enter the following details:
 - Network Name: Hedera Mainnet
 - New RPC URL: <https://mainnet.hashio.io/api>
 - Chain ID: 295
 - Currency Symbol: HBAR
 - Block Explorer URL: <https://hashscan.io/mainnet/>
4. Click Save to add the Hedera Mainnet to MetaMask.

<div align="left">

<figure><figcaption></figcaption></figure>

</div>

<details>

<summary>Alternative RPC Endpoints</summary>

Choose the appropriate endpoint based on whether you want to connect to Mainnet, Testnet, or Previewnet.

Mainnet

<https://mainnet.hashio.io/api>
<https://295.rpc.thirdweb.com>

Testnet

<https://testnet.hashio.io/api>
<https://296.rpc.thirdweb.com>

Previewnet

<https://previewnet.hashio.io/api>
<https://297.rpc.thirdweb.com>

</details>

{% hint style="info" %}

Note: Hashio is currently in beta & for testing purposes only.

{% endhint %}

3. ChainList Method

Alternatively, connect MetaMask to Hedera using ChainList:

1. Go to <https://chainlist.org/?search=hedera>.
2. Click Connect Wallet.
3. Choose your account to connect, click Next, and Connect.
4. Select a network (Hedera Mainnet, Testnet, Previewnet, Localnet), click Add to MetaMask, and then click Approve.

<figure><figcaption></figcaption></figure>

<p>Writer: Krystal, Technical Writer</p> <p>https://github.com/theekrystallee https://x.com/theekrystallee https://hashnode.com/@theekrystallee</p>
<p>Editor: Brendan, DevRel Engineer</p> <p>https://github.com/bguiz https://blog.bguiz.com/</p>

how-to-create-a-personal-access-token-api-key-on-the-hedera-portal.md:

How to Create a Personal Access Token (API Key) on the Hedera Portal

In the Hedera Portal, the Personal Access Token (API key) is a new feature for seamless application development and management. This feature facilitates the process of account recreation and querying for account ID after a test network (Testnet or Previewnet) resets. Whenever the testnet is reset, developers have to manually log in to the portal, recreate an account, get the account ID, and input the new account ID into their development environment. Utilizing this new feature to generate an API key eliminates the need to manually recreate accounts. There are no prerequisites for this tutorial.

By the end of this tutorial, you will know how to:

Create a Hedera Portal profile
Create a new token/API key
Make API calls

Step 1: Create a Hedera Portal Profile

To create your Hedera Portal profile, register here and follow the instructions to complete your profile. Once you've completed setting up your profile, select the test network (Testnet or Previewnet) from the network drop-down menu and create an account.

{% hint style="info" %}

Note: For the purposes of this tutorial, we will be using the Hedera Testnet to demonstrate token creation, but the steps to create tokens for both Testnet and Previewnet are identical.

{% endhint %}

Step 2: Create a New Personal Access Token


Navigate to the Hedera Portal token creation page by clicking on the profile icon in the top-right corner of the portal dashboard and select Personal Access Tokens from the dropdown menu. This page is dedicated to the creation and management of your personal access tokens.

<figure><figcaption></figcaption></figure>

Once you're on the token management page, you will find the option to create a new token. Start by typing in a description for your token in the Description field. This description should ideally reflect the intended use or scope of the token. Click the <mark style="background-color:purple;">CREATE TOKEN</mark> button after entering the description.

<figure><figcaption><p>https://portal.hedera.com/tokens</p></figcaption></figure>

{% hint style="info" %}

 Note: Once your Personal Access Token (API key) is generated, please copy and securely save it in a location where only you have access. Be aware that once you navigate away from the page, the API key will no longer be visible. If you do not save your key, you will need to create a new one, as there is no way to retrieve it later.

{% endhint %}

Step 3: API Calls to List and Get Accounts

Copy your new API token key and save it in a secure place where only you can access it. This key will be used in the API calls below.

<figure><figcaption></figcaption></figure>

API calls using curl

Either of these APIs will trigger the account to be recreated in the background

if the account does not exist (due to a testnet reset).

<details>

<summary>List accounts</summary>

{% code overflow="wrap" %}

bash

```
curl https://portal.hedera.com/api/account -H "Authorization: Bearer <YOUR  
GENERATED API TOKEN>"
```

{% endcode %}

Example API call

{% code overflow="wrap" %}

bash

```
curl https://portal.hedera.com/api/account -H "Authorization: Bearer  
v4.public.eyJzdWIiOiI1M2RlMmI0MC1iOTNiLTExZWUtODk4NC1iYjdkMDE2NTU2ZGQiLCJpYXQiOi  
IyMDI0LTAxLTlYVDE1OjMyOjA2Ljk1MloiLCJqdGkiOiI2Njg4Nzc4Mi1iOTNiLTExZWUtYmRmYi05Zm  
EzMWI3Yjc0ZGIifYer-H5VVjpQloP2U4qwUBBSsb-SQFEYNSrr9-  
8pqsdouDeAp00AWeDre8eGKLEt32JfaQiJ8UrcJyTlwbLfiwk"
```

{% endcode %}

</details>

<details>

<summary>Get account by public key</summary>

{% code overflow="wrap" %}

bash

```
curl https://portal.hedera.com/api/account/<YOUR PUBLIC KEY> -H "Authorization:  
Bearer <YOUR GENERATED API TOKEN>"
```

{% endcode %}

Example API call

{% code overflow="wrap" %}

bash

curl

```
https://portal.hedera.com/api/account/302d300706052b8104000a03220003aaec818ba60d  
7f4e259319804317820f7f4aba3d0048a2f43573d0bbfe9a2254 -H "Authorization: Bearer  
v4.public.eyJzdWIiOiI1M2RlMmI0MC1iOTNiLTExZWUtODk4NC1iYjdkMDE2NTU2ZGQiLCJpYXQiOi  
IyMDI0LTAxLTlYVDE1OjMyOjA2Ljk1MloiLCJqdGkiOiI2Njg4Nzc4Mi1iOTNiLTExZWUtYmRmYi05Zm  
EzMWI3Yjc0ZGIifYer-H5VVjpQloP2U4qwUBBSsb-SQFEYNSrr9-  
8pqsdouDeAp00AWeDre8eGKLEt32JfaQiJ8UrcJyTlwbLfiwk"
```

{% endcode %}

</details>

Example API call response

{% code overflow="wrap" %}

json

{"accounts":

```
[{"accountNum": "7685875", "shard": "0", "realm": "0", "balanceLimit": "100000000000", "  
network": "testnet", "keyType": "ecdsa", "lastDisbursementAt": "2024-01-22  
15:29:40.004689+00", "publicKey": "302d300706052b8104000a03220003aaec818ba60d7f4e2  
59319804317820f7f4aba3d0048a2f43573d0bbfe9a2254", "privateKey": "302d300706052b810  
4000a03220003aaec818ba60d7f4e259319804317820f7f4aba3d0048a2f43573d0bbfe9a2254", "
```

```
scheduledForDisbursement":false,"balance":{"valueInTinybar":"1000000000000"}}}]
{% endcode %}
```

The data returned by the `curl` command is a JSON object detailing information about the accounts on the Hedera network. Here's a breakdown of each field and what it represents:

accountNum: Account identifier, e.g., "7685875".
 shard: Shard ID, part of the account address, e.g., "0".
 realm: Realm ID within the shard, e.g., "0".
 balanceLimit: Maximum account balance, e.g., "1000000000000".
 network: Network type - mainnet, testnet, previewnet.
 keyType: Cryptographic key type (ECDSA, ED25519), here "ecdsa".
 lastDisbursementAt: Last disbursement timestamp, e.g., "2024-01-22
 15:29:40.004689+00".
 publicKey: Account's public key in hexadecimal.
 privateKey: Account's private key in hexadecimal.
 scheduledForDisbursement: Indicates if disbursement is scheduled, here false.
 balance: Contains valueInTinybar, the account balance in tinybar units, e.g.,
 "1000000000000".

Writer: Krystal, Technical Writer	
https://github.com/theekrystallee	Twitter
https://github.com/theekrystallee	https://github.com/theekrystallee
Editor: Simi, Sr. Software Manager	
https://github.com/SimiHunjan	https://www.linkedin.com/in/shunjan/

```
# how-to-generate-a-random-number-on-hedera.md:
```

How to Generate a Random Number on Hedera

Pseudorandom numbers are used in applications like lotteries, gaming, and even random selection in NFT sales/giveaways. In some cases, it is necessary to prove to third parties that a random number was generated without the influence or control of a user/application.

With Hedera, you have a transaction and Solidity library available to generate pseudorandom numbers with just a few lines of code. This means that when your application generates a random number, anyone can verify that the number was truly generated by the Hedera network - without being influenced by any one user.

In this tutorial, you will learn how to generate random numbers on Hedera using the JavaScript SDK and Solidity. Keep in mind that the random number generator covered here is secure enough for practical applications, but you should carefully consider whether it is secure enough for your specific purpose. For details on limitations and security considerations, see HIP-351 (special mention and thanks to LG Electronics for the help and input in formulating this Hedera Improvement Proposal).

This resource is helpful if you want to learn more about the difference between pseudorandom and random numbers. For simplicity, we'll use both terms interchangeably throughout this tutorial.

Prerequisites

Get a Hedera Testnet account [here](#).
Get the example code from [GitHub](#).
Set up your environment and create a client [here](#).


 **Note:** This tutorial requires the use of the following tools: Hedera JavaScript SDK, Solidity with libraries for random number generation (HIP-351), and the Mirror Node REST API ([learn more](#)) and explorer ([HashScan](#)).

Table of Contents

1. Generate Random Numbers (SDK)
2. Generate Random Numbers (Solidity)
 1. Deploy Contract
 2. Execute Contract
 3. Query Results
3. Summary
4. Additional Resources

Generate Random Numbers Using the SDK

Step 1: The Operator is the only account involved in submitting transactions to the Hedera network. Your testnet credentials from the Hedera portal should be used for the operator variables, which are used to initialize the Hedera client that submits transactions to the network and gets confirmations.

The constants `lo` and `hi` are the lower and upper limits for the random number, respectively

Generate a random number (`randomNum`) 5 times with a for loop

Use `PrngTransaction()` in the SDK to create a transaction that generates a pseudorandom number

If a positive value is provided to the `setRange()` method, then the transaction record will contain a 32-bit pseudorandom integer that is equal or greater than 0 and less than `hi`

If a range is not specified, then the transaction record contains the 384-bit array of pseudorandom bits

The client is used to execute the transaction and obtain the transaction record
Output to the console the random number for each loop run


```
{% code title="index.js" %}
javascript
// STEP 1 =====
console.log(\nSTEP 1 =====\n);
console.log(- Generating random numbers with the SDK...\n);

const lo = 0;
const hi = 50;

let randomNum = [];
for (var i = 0; i < 5; i++) {
  const randomNumTx = await new
PrngTransaction().setRange(hi).execute(client);
  const randomNumRec = await randomNumTx.getRecord(client);
  randomNum[i] = randomNumRec.prngNumber;
  console.log(- Run #${i + 1}: Random number = ${randomNum[i]});
}
```

```
{% endcode %}
```

```
<details>
```

```
<summary><strong>Console Output 
```

```
STEP 1 =====
```

```
\- Generating random numbers with the SDK...
```

```
\- Run #1: Random number = 10
```

```
\- Run #2: Random number = 7
```

```
\- Run #3: Random number = 14
```

```
\- Run #4: Random number = 44
```

```
\- Run #5: Random number = 27
```

```
</details>
```

Generate Random Numbers Using Solidity

Step 2: Deploy a Solidity smart contract to generate the random number. After completing all steps, your console should look something like this.

Deploy Contract

In the index.js file:

```
Set the gas limit (gasLim) to be 4,000,000 and define the contract bytecode
Deploy the contract using the helper function contracts.deployContractFcn
The function returns the contractId in Hedera format and contractAddress in
Solidity format
The inputs are the bytecode, gasLim, and client
Output to the console contractId and contractAddress
```

You can see the Solidity contract PrngSystemContract in the second tab below. This example calls a precompiled contract with address 0x169. For additional details about the contract and the functions getPseudorandomSeed and getPseudorandomNumber, check out HIP-351. The first function generates a 256-bit pseudorandom seed and returns the corresponding bytes; you can then use that seed to get a random number. The second function operates on those bytes to return the random number. There is also a contract function getNumber which reads the random number from the contract state variable, randNum.

```
{% tabs %}
{% tab title="Index.js" %}
javascript
// STEP 2 =====
console.log(\nSTEP 2 =====\n);
console.log(- Generating random number with Solidity...\n);

// Deploy the Solidity contract
let gasLim = 4000000;
const bytecode = contract.object;
const [contractId, contractAddress] = await
contracts.deployContractFcn(bytecode, gasLim, client);
console.log(- Contract ID: ${contractId});
console.log(- Contract ID in Solidity address format: ${contractAddress});
```

```

{% endtab %}

{% tab title="PrngSystemContract.Sol" %}
javascript
// SPDX-License-Identifier: Apache-2.0
pragma solidity >=0.4.9 <0.9.0;

import "../IPrngSystemContract.sol";

contract PrngSystemContract {
    address constant PRECOMPILEADDRESS = address(0x169);
    uint32 randNum;

    function getPseudorandomSeed() external returns (bytes32 randomBytes) {
        (bool success, bytes memory result) = PRECOMPILEADDRESS.call(
abi.encodeWithSelector(IPrngSystemContract.getPseudorandomSeed.selector));
        require(success);
        randomBytes = abi.decode(result, (bytes32));
    }

    /
    Returns a pseudorandom number in the range lo, hi) using the seed
    generated from "getPseudorandomSeed"
    /
    function getPseudorandomNumber(uint32 lo, uint32 hi) external returns
(uint32) {
        (bool success, bytes memory result) = PRECOMPILEADDRESS.call(
abi.encodeWithSelector(IPrngSystemContract.getPseudorandomSeed.selector));
        require(success);
        uint32 choice;
        assembly {
            choice := mload(add(result, 0x20))
        }
        randNum = lo + (choice % (hi - lo));
        return randNum;
    }

    function getNumber() public view returns (uint32) {
        return randNum;
    }
}

```

```

{% endtab %}
{% endtabs %}

```

Helper Functions:

The function `contracts.deployContractFcn` uses `[ContractCreateFlow()` to store the bytecode and deploy the contract on Hedera. This single call handles for you the operations `FileCreateTransaction()`, `FileAppendTransaction()`, and `ContractCreateTransaction()`. This helper function simplifies the contract deployment process and is reusable in case you need to create more contracts in the future.

```

javascript
export async function deployContractFcn(bytecode, gasLim, client) {
    const contractCreateTx = new
ContractCreateFlow().setBytecode(bytecode).setGas(gasLim);
    const contractCreateSubmit = await contractCreateTx.execute(client);
    const contractCreateRx = await contractCreateSubmit.getReceipt(client);

```

```

    const contractId = contractCreateRx.contractId;
    const contractAddress = contractId.toSolidityAddress();
    return [contractId, contractAddress];
}

```

Execute Contract

Use the helper `contracts.executeContractFcn` to execute the contract function `getPseudorandomNumber`

Use `ContractFunctionParameters()` from the SDK to specify the parameters for the contract function (`randNumParams`). Pass the lower (`lo`) and upper (`hi`) limits for the random number

The inputs are the contract ID (`contractId`), the contract function to execute, `randNumParams`, `gasLim`, and `client`

The contract function calls a precompiled contract and gets the bytes for the random seed. The random number is calculated, stored in the state variable `randNum`, and returned by the function

The helper function returns the record object of the transaction (`randNumRec`), which is used to obtain the status of the transaction

Output to the console:

The status of the contract call

```

javascript
// Execute the contract
const randNumParams = new
ContractFunctionParameters().addUint32(lo).addUint32(hi);
const randNumRec = await contracts.executeContractFcn(contractId,
"getPseudorandomNumber", randNumParams, gasLim, client);
console.log(- Contract execution: ${randNumRec.receipt.status} \n);

```

Helper Functions:

The function `contracts.executeContractFcn` uses `ContractExecuteTransaction()` in the SDK to call the specified contract function.

```

javascript
export async function executeContractFcn(cId, fcnName, params, gasLim, client) {
    const contractExecuteTx = new
ContractExecuteTransaction().setContractId(cId).setGas(gasLim).setFunction(fcnName,
params);
    const contractExecuteSubmit = await contractExecuteTx.execute(client);
    const contractExecuteRec = await contractExecuteSubmit.getRecord(client);
    return contractExecuteRec;
}

```

Query Results

You will learn various ways to obtain the random number from the Solidity contract. The best approach depends on your use case and preference. You can get the random number by: using a transaction record, doing a contract call to read state variables, and checking a mirror node explorer.

Use the helper function `queries.txRecQueryFcn` to obtain information from the transaction record

The function returns a record query object (`recQuery`)

The inputs are the ID of the relevant transaction from the record object (`randNumRec.transactionId`) and `client`

The query of the transaction record, `recQuery`, was configured to return information on child transactions. Thus, the first child transaction of the contract execution (parent transaction) contains information about the bytes needed for the random number – see `recQuery.children\[0\].prngBytes`

The random number is obtained from doing a modulo operation (%) of the integer value of the last four bytes and the specified range

Use the helper contracts.callContractFcn to call the contract function getNumber, which reads the random number from a state variable in the contract

The inputs are contractId, the contract function to call, gasLim, and client

The helper function returns randNumResult, which is used to obtain the random number

Use the helper function queries.mirrorTxQueryFcn to obtain transaction information from the mirror nodes

The function returns a mirror node REST API request about the relevant transaction (randNumInfo) and a string with a mirror explorer URL (randNumExpUrl)

The input is the ID of the relevant transaction from the record object (randNumRec.transactionId)

Output to the console:

The random number obtained using both the transaction record and using the contract function

A message indicating if the random number obtained with the two methods above matches or not

The mirror node explorer URL with more details about the transaction

```
javascript
// Query the transaction record to get the random number from bytes
const recQuery = await queries.txRecQueryFcn(randNumRec.transactionId, client);

let lowOrderBytes = new Uint8Array(recQuery.children[0].prngBytes).slice(28, 32);
let dataview = new DataView(lowOrderBytes.buffer);
let range = hi - lo;
let int32be = dataview.getUint32(0);
let randNum = int32be % range;
console.log(- The random number (using transaction record) = ${randNum});

// Call the contract to read random number using the getNumber function
const randNumResult = await contracts.callContractFcn(contractId, "getNumber", gasLim, client);
console.log(- The random number (using contract function) = ${randNumResult.getUint32(0)});
randNum === randNumResult.getUint32(0) ? console.log(- The random number checks out ✅) : console.log(- Random number doesn't match ❌);

// Check a Mirror Node Explorer
const [randNumInfo, randNumExpUrl] = await queries.mirrorTxQueryFcn(randNumRec.transactionId);
console.log(\n- See details in mirror node explorer: \n${randNumExpUrl});

console.log(
=====
THE END - NOW JOIN: https://hedera.com/discord
=====\\n);
}
```

Helper Functions

The function queries.txRecordQueryFcn uses TransactionRecordQuery() in the SDK to obtain details about the contract execution transaction. Note that the .setIncludeChildren method is set to true to get information about all the children transactions under the contract execution – this includes the transaction generating the random number.

The function contracts.callContractFcn uses ContractCallQuery() in the SDK to call a contract function that reads a state variable containing the random number.

The function `queries.mirrorTxQueryFcn` obtains transaction information from the mirror nodes. The function introduces a delay of 10 seconds to allow for the propagation of information to the mirror nodes. It then formats the transaction ID and performs string operations to return a mirror REST API query and a mirror node explorer URL.

```
{% tabs %}
{% tab title="Queries.TxRecordQueryFcn" %}
javascript
export async function txRecQueryFcn(txId, client) {
  const recQuery = await new
TransactionRecordQuery().setTransactionId(txId).setIncludeChildren(true).execute
(client);
  return recQuery;
}

{% endtab %}

{% tab title="Contracts.CallContractFcn" %}
javascript
export async function callContractFcn(cId, fcnName, gasLim, client) {
  const contractCallTx = new
ContractCallQuery().setContractId(cId).setGas(gasLim).setFunction(fcnName);
  const contractCallSubmit = await contractCallTx.execute(client);
  return contractCallSubmit;
}

{% endtab %}

{% tab title="Queries.MirrorTxQueryFcn" %}
javascript
export async function mirrorTxQueryFcn(txIdRaw) {
  // Query a mirror node for information about the transaction
  const delay = (ms) => new Promise((res) => setTimeout(res, ms));
  await delay(10000); // Wait for 10 seconds before querying a mirror node

  const txIdPretty = prettify(txIdRaw.toString());
  const mirrorNodeExplorerUrl = https://hashscan.io/testnet/transaction/$
{txIdPretty};
  const mirrorNodeRestApi =
https://testnet.mirrornode.hedera.com/api/v1/transactions/${txIdPretty};
  let mQuery = [];
  try {
    mQuery = await axios.get(mirrorNodeRestApi);
  } catch {}
  return [mQuery, mirrorNodeExplorerUrl];
}

function prettify(txIdRaw) {
  const a = txIdRaw.split("@");
  const b = a[1].split(".");
  return `${a[0]}-${b[0]}-${b[1]}`;
}

{% endtab %}
{% endtabs %}
```

<details>

<summary>Console Output ☒

STEP 2 =====


The Hedera Local Node project enables developers to establish their own local network for development and testing. The local network comprises the consensus node, mirror node, JSON-RPC relay, and other Hedera products, and can be set up using the CLI tool and Docker. This setup allows you to seamlessly build and deploy smart contracts from your local environment.

By the end of this tutorial, you'll be equipped to run a Hedera local node and generate keys, allowing you to test your projects and deploy projects in your local environment.

Prerequisites

Node.js >= v14.x
NPM >= v6.14.17\\
Minimum 16GB RAM
Docker >= v20.10.x
Docker Compose >= v2.12.2
Have Docker running on your machine with the correct configurations.

<details>

<summary>Docker configuration </summary>

Ensure the VirtioFS file sharing implementation is enabled in the docker settings.

Ensure the following configurations are set at minimum in Docker Settings -> Resources and are available for use:

CPUs: 6
Memory: 8GB
Swap: 1 GB
Disk Image Size: 64 GB

Ensure the Allow the default Docker sockets to be used (requires password) is enabled in Docker Settings -> Advanced.

Note: The image may look different if you are on a different version

</details>

\\Local node can be run using Docker or NPM but we will use Docker for this tutorial. Here are the installation steps for NPM.

Table of Contents

1. Start Your Local Network
2. Generate Keys
3. Stop Your Local Network
4. Additional Resources

Start Your Local Network

Open a new terminal and navigate to your preferred directory where your Hedera Local Node project will live. Run the following command to clone the repo and install dependencies to your local machine:


```
bash
git clone https://github.com/hashgraph/hedera-local-node.git
cd hedera-local-node
npm install
```

For Windows users: You will need to update the file endings of
compose-network/mirror-node/init.sh by running this in WSL:

```
bash
dos2unix compose-network/mirror-node/init.sh
```

Ensure Docker is installed and open on your machine before running this command
to get the network up and running:

```
bash
// starts and generates the first 30 accounts
npm run start -- -d
```

or

```
// will start local node but will not generate the first 30 accounts
docker compose up -d
```

Generate Keys

To generate accounts with random private keys, run the generate-accounts
command. Specify the number of accounts generated by appending the number to the
hedera generate-account command. For example, to generate 5 accounts, run hedera
generate-accounts 5.

<details>

<summary><code>hedera generate-accounts 5</code> </summary>

Generating accounts in synchronous mode...

```
|-----|
|-----|
|-----| Accounts list ( ECDSA  keys)
|-----|
|-----|
|-----|
| id | private key |
balance |
|-----|
|-----|
| 0.0.1033 - 0xc3d34a00d3ffff542e350a5e61cb41509812bf23ea581f83a0a862c94d8c69704
- 10000 ¢ |
| 0.0.1034 - 0xa4189ab682ba43925ce654ca09800bba86cf8b1b7f889006d5170d95f4fed365
- 10000 ¢ |
| 0.0.1035 - 0xf9106e9841677136c9cbe8c114dab80470ca62a15bfe9c777006bcb114288c22
- 10000 ¢ |
| 0.0.1036 - 0xe3517a9235971be1e1f95e791f3ffd7d753a652799fa11f1ace626036c4db275
- 10000 ¢ |
| 0.0.1037 - 0x636926cf2f6f9fd0a58043c600390eeef0bbbed9d4b8a113ea68a8d67f922d04e
- 10000 ¢ |
|-----|
|-----|
```

```

|-----|
|-----|
|-----| Accounts list (Alias ECDSA
keys) |-----|
|-----|
| id | public address |
private key | balance |
|-----|
| 0.0.1038 - 0xaBE90e20f394629e054Bc1E8F1338Fe8ea94F0b5 -
0x444913bd258f764e62db6c87abde7ca52ec22985db8c91b8c3b2b4f2c51775f0 - 10000 h |
| 0.0.1039 - 0x26d941d8E1f6bF9B0F7e5156fA6ff02acEd0DF3E -
0xea25f427caf7029989669f93926b7902dde5361b176b4bc17b8ec0a967beaa0b - 10000 h |
| 0.0.1040 - 0x64001c2d1f3a8d3574435B4F125944018E2E584D -
0xf2deb678a1e67e288d8a128334f41c890e7600b2a5471ecc9a3af4824e3021b7 - 10000 h |
| 0.0.1041 - 0x6bE22CD9D16b64969683B74897E4EBB30c7c30E8 -
0xb9c2480cddbddd2ecd6e032b87820c29e8791ad4f53b89f829269d856c835819 - 10000 h |
| 0.0.1042 - 0x992d8aD211b28B23589c0b3Fe30de6C90662C4aB -
0x7e8bb0d85a8d80fa2eb2c9f6bd5c9b1a2c2f9f6992c7fffd201c8e81f0ec0000 - 10000 h |
|-----|
|-----|
|-----|
|-----| Accounts list (ED25519 keys)
|-----|
|-----|
| id | private key |
balance |
|-----|
| 0.0.1043 - 0xd4917e152ca922b8bfba9ffc3486512ae25ec0a75b05c44f517b11cd12fd949b
- 10000 h |
| 0.0.1044 - 0xbaeec69382fbb43e4d521b3d8717c9cba610a1fbcaededaaf4408c3138a683ae
- 10000 h |
| 0.0.1045 - 0x1f5c4b2efd3c36d29e9d2e16a825abd001f99bff2388bb8c6011cd5f956023c9
- 10000 h |
| 0.0.1046 - 0x1976acdd5e71ce7e8db4cb0aa112fa1c16876155f0f20b9b7029916073f1d67f
- 10000 h |
| 0.0.1047 - 0x6e29f48b11ffc77e277f0500d607b35956da58f1ed30aad003fb1846bfff483
- 10000 h |
|-----|
|-----|

```

</details>

```
{% hint style="info" %}
```

Please note: Since the first 10 accounts generated are with predefined private keys, if you need 5 generated with random keys, you will run hedera start 15. The same rule applies when you use the hedera generate-accounts command.

```
{% endhint %}
```

Grab any of the account private keys generated from the Alias ECDSA keys Accounts list. This will be used as the LOCALNODEOPERATORPRIVATEKEY environment variable value in your .env file of your project.

Stop Your Local Network

To stop your local node, you can run the hedera stop command. If you want to

keep any files created manually in the working directory, please save them before executing this command.

<details>

<summary><code>hedera stop</code></summary>

Stopping the network...
Stopping the docker containers...
Cleaning the volumes and temp files...

</details>

Alternatively, run `docker compose down -v; git clean -xfd; git reset --hard` to stop the local node and reset it to its original state.

<details>

<summary><code>docker compose down -v; git clean -xfd; git reset --hard</code></summary>

bash

[+] Running 27/27

✓ Container mirror-node-web3	Removed	3.5s
✓ Container json-rpc-relay-ws	Removed	10.8s
✓ Container mirror-node-monitor	Removed	3.7s
✓ Container relay-cache	Removed	0.9s
✓ Container prometheus	Removed	0.9s
✓ Container record-sidecar-uploader	Removed	0.0s
✓ Container grafana	Removed	0.9s
✓ Container hedera-explorer	Removed	10.4s
✓ Container json-rpc-relay	Removed	10.7s
✓ Container account-balances-uploader	Removed	0.1s
✓ Container envoy-proxy	Removed	1.0s
✓ Container mirror-node-grpc	Removed	2.7s
✓ Container mirror-node-rest	Removed	10.4s
✓ Container network-node	Removed	10.8s
✓ Container mirror-node-importer	Removed	10.4s
✓ Container record-streams-uploader	Removed	0.0s
✓ Container haveged	Removed	0.0s
✓ Container mirror-node-db	Removed	0.3s
✓ Container minio	Removed	0.0s
✓ Volume prometheus-data	Removed	0.0s
✓ Volume minio-data	Removed	0.0s
✓ Volume mirror-node-postgres	Removed	0.1s
✓ Volume grafana-data	Removed	0.2s
✓ Network network-node-bridge	Removed	0.1s
✓ Network hedera-local-nodedefault	Removed	0.2s
✓ Network cloud-storage	Removed	0.2s
✓ Network mirror-node	Removed	0.2s

Removing .husky//

Removing network-logs/

Removing nodemodules/

HEAD is now at

</details>

🔔 Note: All available commands can be checked out [here](#).

Additional Resources

- ☞ Hedera Local Node Repository
- ☞ Hedera Local Node CLI Tool Commands
- ☞ Hedera Local Node Docker Setup \[Video Tutorial]

javascript-testing.md:

description: How to test Javascript, using Javascript

JavaScript Testing

Developing DApps typically involves using quite a lot of JavaScript. This can happen in several different areas:

- The DApp client itself, when the DApp is browser-based
- The backend APIs, when the DApp uses some web2 server components
- Test code for smart contracts
- Test code for DApp client
- Test code for backend APIs

Looking at the latter three, suffice it to say that learning testing with JavaScript is a fundamental skill for DApp developers. Unfortunately, it is one that is often overlooked. This tutorial aims to fill that gap. If you're planning to create a DApp on Hedera, but wish to brush up on the basics of testing first, start here!

What we will cover

A test runner is a developer tool that helps you to:

- Structure your tests.
- Execute your tests against a target application (known as the system under test).
- Report the results of the tests.

One of the most popular JavaScript test runners is Mocha; and this is what you will be using in this tutorial. We picked this because both Hardhat and Truffle, two of the most popular smart contract development frameworks, use Mocha under the hood in their built-in smart contract testing features. This means that the syntax you learn here will be familiar and useful when building smart contracts for the Hedera Smart Contract Service (HSCS).

In this tutorial, let's start with a very simple application and modify its implementation to cover four different scenarios you'll encounter during development.

- True positive
- True negative
- False negative
- False positive

<details>

<summary>Definitions of these test outcome scenarios</summary>

We'll define these as we go through the tutorial steps, but in case you would like a preview:

True positive -> both the implementation and specification are correct.
True negative -> the implementation is wrong, but the specification is correct.
False negative -> the implementation is correct, but the specification is wrong.
False positive -> both the implementation and specification are wrong.

</details>

For each scenario, you'll cover what to look out for and how to handle it properly.

Let's begin!

Prerequisites

Prior knowledge

JavaScript syntax

System

git installed

Minimum version: 2.37

Recommended setup method: Install Git (Github docs)

SSH keys configured for Github: Add SSH key (Github docs)

NodeJS + npm installed

Minimum version (NodeJS): 18

Recommended setup method for Linux & Mac: nvm

Recommended setup method for Windows: nvm-windows

Step 1: Set up the project

This has already been (mostly) done. All that's left for you to do is clone the accompanying tutorial GitHub repository and install the dependencies:

```
sh
git clone git@github.com:hedera-dev/js-testing.git
cd js-testing
npm install
```

{% hint style="info" %}

If you do not have SSH available on your system, or are unable to configure it for GitHub, you may wish to try this git command instead:\

```
\
git clone https://github.com/hedera-dev/js-testing.git
{% endhint %}
```

Open this repository in your code editor, and you'll find the following files:

package.json

mocha is installed as a devDependency.

The test command is configured to run mocha on all files with a .spec.js file extension.

The test:generative command is configured to run mocha on all files with a .genspec.js file extension - note that this is only needed for the bonus step.

my-app.js

The system under test, also referred to as the implementation.
This will contain a single add function - very simple, but enough to demonstrate a point.

my-app.spec.js

The tests, also referred to as the specification.
These specify what the behavior of the implementation should be.

Step 2: Implement the system under test

In the add function within my-app.js, you should see a comment marking Step 2.
It looks like this:

```
javascript
// TODO: Implement the system under test
// Follow step (2) in tutorial to complete the following section.
```

This is where you'll add the code for this step. The implementation for addition is self-explanatory.

```
javascript
return x + y;
```

It should now look like this:

```
javascript
// TODO: Implement the system under test
// Follow step (2) in tutorial to complete the following section.
return x + y;
```

```
{% hint style="info" %}
```

Note: In the subsequent steps of this tutorial, you will follow the same pattern as above. However, you will not copy the comments marking the steps for the remainder of the tutorial and instead only include the new/changed lines of code.

```
{% endhint %}
```

Now you have completed your system under test!

Step 3: Implement the tests

In my-app.spec.js, find the test block with the title works with known values, and add the following implementation where indicated with the comment.

```
javascript
const actual = add(1, 2);
const expected = 3;
assert.equal(actual, expected);
```

The actual value is obtained from invoking the system under test.

The expected value is simply written by you as the test writer.

Finally, you have an assertion that checks that the actual and expected values match.

Now you have completed the specification!

Since both the system under test and the tests for it are ready, you're ready to invoke the test runner.

```
bash
npm run test
```

You should now see some output that looks similar to the following:

```
bash
> tutorial-js-testing@0.0.0 test
> mocha './.spec.js'

my-app
  add
    ✓ works with known values
    ✓ is associative with known values
    ✓ is commutative with known values

3 passing (3ms)
```

So all the tests have passed: One that you have just added ('known values'), and two more that were included in the initial state of the tutorial repo ('associative', and 'commutative').

You could wrap up here... but you're not quite done yet. This is a true positive scenario, where both the implementation and the specification are correct. But there are three other possible scenarios that you're likely to encounter when writing tests, so let's go through them in the next few steps!

Step 4: Switch to a true negative scenario

In the true negative scenario, the implementation is wrong, and the specification is correct. This is probably the most common test failure scenario you'll encounter during development.

In my-app.js, comment out the existing code from Step 2, and add this new implementation for Step 4.

```
javascript
return x + y + 1;
```

In my-app.spec.js, comment out the assertion from Step 3 (keep the other 2 lines of code), and add this new line of code for Step 4. (Note that this happens to be the same as before.)

```
javascript
assert.equal(actual, expected);
```

Now let's re-run the tests.

```
bash
npm run test
```

You should now see some output that looks similar to the following:

```

bash
> tutorial-js-testing@0.0.0 test
> mocha './.spec.js'

my-app
  add
    1) works with known values
      ✓ is associative with known values
      ✓ is commutative with known values

  2 passing (4ms)
  1 failing

  1) my-app
       add
         works with known values:

      AssertionError [ERR_ASSERTION]: 4 == 3
      + expected - actual

      -4
      +3

```

This shows that the actual value was 4, while the expected value was 3. While this is an error, it is actually a good thing - the main point of writing tests is to catch errors in your application, and that is precisely what has happened here. Even though this is a contrived example, as the error in the implementation is so obvious, it nevertheless illustrates the developer workflow involved: Finding bugs in the implementation of an application by specifying how it should behave, and using a test runner to detect where there are problems.

```

{% hint style="info" %}
Interestingly, the tests for associativity and commutativity do not fail, even
with this bug in the implementation. This illustrates that not all tests are
able to catch bugs, and why several different tests are necessary. This can be
the case even for simple/ "obvious" bugs.
{% endhint %}

```

As a developer, at this point, you would typically fix the error in the implementation, and then re-run the tests to ensure that they pass once again. You will do so eventually, towards the end of this tutorial. However, for now, you'll move on to another scenario.

Step 5: Switch to a false negative scenario

In the false negative scenario, the implementation is correct, and the specification is wrong. This is not as common of a scenario that you'll encounter during development as the false positive, but it is nonetheless important to be able to recognize it when it does occur and rectify it accordingly.

In `my-app.js`, comment out the existing code from Step 4, and add this new implementation for Step 5.

```

javascript
return x + y;

```

Likewise, in `my-app.spec.js`, comment out the existing code from Step 4, and add this new implementation for Step 5.


```
javascript
assert.equal(actual, expected + 1);
```

Now let's re-run the tests.

```
sh
npm run test
```

You should now see some output that looks similar to the following:

```
bash
> tutorial-js-testing@0.0.0 test
> mocha './.spec.js'

my-app
  add
    1) works with known values
      ✓ is associative with known values
      ✓ is commutative with known values

  2 passing (4ms)
  1 failing

  1) my-app
       add
         works with known values:

      AssertionError [ERR_ASSERTION]: 3 == 4
      + expected - actual

      -3
      +4
```

This time, the error shows that the actual value was 3, while the expected value was 4. This is the opposite of previous test error - the actual and expected values have swapped positions. The somewhat tricky thing here is to not "default" to assuming that the error must be in the implementation, and therefore only look into finding a bug in the implementation. The correct thing to do, whenever the test runner reports a test failure is to analyse both the implementation and specification, and identify which one of them contains an error.

As a developer, at this point, you would typically fix the error in the specification, and then re-run the tests to see if they pass once again. Instead, let's move on to another scenario.

Step 6: Switch to a false positive scenario

In the false positive scenario, the implementation is wrong, and the specification is also wrong. This is typically the least common scenario that you'll encounter during development. And it can be extremely tricky to even identify, as you'll see shortly.

In my-app.js, comment out the existing code from Step 5, and add this new implementation for Step 6.

```
javascript
return x + y + 1;
```

Likewise, in `my-app.spec.js`, comment out the existing code from Step 5, and add this new implementation for Step 6.

```
javascript
assert.equal(actual, expected + 1);
```

Now let's re-run the tests.

```
sh
npm run test
```

You should now see some output that looks similar to the following:

```
bash
> tutorial-js-testing@0.0.0 test
> mocha './.spec.js'

my-app
  add
    ✓ works with known values
    ✓ is associative with known values
    ✓ is commutative with known values

3 passing (2ms)
```

Somewhat surprising is it not? Even though both the implementation and specification were wrong, all the tests pass. In fact, the output from the test runner looks identical to when both the implementation and specification were correct. This occurs because the implementation code and the specification code coincidentally happen to make the same error, and they "cancel" each other out.

While in this case it may be obvious, that is only because of the simplicity of this tutorial. Spotting false positive scenarios in a complex application is much harder. More than just tricky, this scenario is insidious due to its ability to have bugs and yet pass the tests meant to catch them. In fact, there is no way for a developer to identify a false positive scenario from the test results alone.

```
{% hint style="info" %}
```

False positive scenarios are typically spotted through manual reviews of the both the implementation and specification code, or other code quality processes such as static/ dynamic analyses and code coverage. These are beyond the scope of this tutorial.

```
{% endhint %}
```

Now let's finally fix the code, and go back to the true positive scenario, where you started off.

Step 7: Switch back to a true positive scenario

In the true positive scenario, the implementation is correct, and the specification is correct as well. This is the ideal scenario among the 4 possible ones that you've covered thus far. Whenever there are errors identified in either the implementation or specification, the goal is to fix them such that you return to this true positive scenario.

In `my-app.js`, comment out the existing code from Step 6, and add this new

implementation for Step 7.

```
javascript
return x + y;
```

Likewise, in `my-app.spec.js`, comment out the existing code from Step 6, and add this new implementation for Step 7.

```
javascript
assert.equal(actual, expected);
```

Now let's re-run the tests.

```
sh
npm run test
```

You should now see some output that looks similar to the following:

```
bash
> tutorial-js-testing@0.0.0 test
> mocha './.spec.js'

my-app
  add
    ✓ works with known values
    ✓ is associative with known values
    ✓ is commutative with known values

  3 passing (2ms)
```

Back to all tests passing (for the right reasons)!

Step 8: Bonus - Add generative testing

Thus far, all the tests that you have written (in `my-app.spec.js`) are example based tests. Essentially your tests consist of one or more interactions with the system under test actual value, an expected value, and an assertion that the actual value matches the expected value. In this step, we'll add a different type of tests to this project: Generative testing.

```
{% hint style="info" %}
Generative testing may be thought of as an abstract form of example based
testing, where you write the tests in such a way that they do not require you to
specify any input values for the examples that you write tests for. Instead you
let the test runner generate value at random as the input value used in your
tests. In other words, the tested examples are generated by the test runner,
instead of being manually specified.
{% endhint %}
```

The demo repo for this tutorial has already been wired up with a dependency named `testcheck.js`. This provides a plugin that augments Mocha with a new `check.it()` function that behaves in a very similar manner to `it()` in Mocha. The key difference being that it contains some generator functions that produce random values of a specified type. You will be using `gen.int` to produce random Integer values to use as inputs for the parameters of the `add()` function the application.

Checkout the `my-app.genspec.js` file, where this has already been set up for you.

The 'commutative' and 'associative' tests have been copied over from my-app.spec.js, and modified to use check.it() plus gen.int. Compare the method signatures of the callback functions, and the difference between example based testing and generative testing should be clear.

In my-app.genspec.js, replace these two lines of code for Step 8, in the 'is associative' test.

```
javascript
const lhsOfEquation = add(a, add(b, c));
const rhsOfEquation = add(add(a, b), c);
```

Also, in the 'is commutative' test, replace these two lines of code.

```
javascript
const lhsOfEquation = add(a, b);
const rhsOfEquation = add(b, a);
```

Note that in both tests, the assertions do not need to be modified, you can keep them as is.

```
{% hint style="info" %}
Compare this to the similar example based tests for associativity and
commutativity within my-app.spec.js. These will use the same literal/ hard-coded
values every time you run the tests, unlike the generative tests within my-
app.genspec.js.
```

Note that the values of a, b, and c above are the randomly generated integers. This means that every time you run the generative tests, you will be testing the add() function using a new set of input values.

```
{% endhint %}
```

Now let's run the new generative tests.

```
sh
npm run test:generative
```

You should now see some output that looks similar to the following:

```
bash
> tutorial-js-testing@0.0.0 test:generative
> mocha './.genspec.js'
```

```
my-app
  add
    ✓ is associative
    ✓ is commutative
bas
  2 passing (17ms)
```

These generative tests are passing as well and form an additional layer of verification of the system under test. You can now be extra sure that the implementation is indeed correct.

Congrats!

You've completed this tutorial! :tada: :tada: :tada:

Now that you have covered the basics of testing, you're ready to test more complex applications. If you have smart contracts that you intend to deploy to

Hedera Smart Contract Service (HSCS), it is best practice to test them before deployment. If you use Hardhat or Truffle, you will already be familiar with much of the syntax for the specification code.

```
<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Brendan, DevRel Engineer</p><p><a href="https://github.com/bguiz">GitHub</a> | <a href="https://blog.bguiz.com">Blog</a></p></td><td align="center"><p>Editor: Abi Castro, DevRel Engineer</p><p><a href="https://github.com/a-ridley">GitHub</a> | <a href="https://twitter.com/ridley">Twitter</a></p></td><td align="center"><p><a href="https://twitter.com/ridley">https://twitter.com/ridley</a></p></td></tr></tbody></table>
```

README.md:

More Tutorials

```
<table data-view="cards"><thead><tr><th align="center"></th><th data-hidden></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><mark style="color:purple;"><strong>THE BASICS</strong></mark></td><td></td><td></td><td><a href="./#the-basics">#the-basics</a></td></tr><tr><td align="center"><mark style="color:purple;"><strong>SMART CONTRACTS</strong></mark></td><td></td><td></td><td><a href="./#smart-contracts">#smart-contracts</a></td></tr><tr><td align="center"><mark style="color:purple;"><strong>NFTs &#x26; FUNGIBLE TOKENS</strong></mark></td><td></td><td></td><td><a href="./#nfts-and-fungible-tokens-on-hedera">#nfts-and-fungible-tokens-on-hedera</a></td></tr><tr><td align="center"><mark style="color:purple;"><strong>ACCOUNTS, KEYS, HBAR</strong></mark></td><td></td><td></td><td><a href="./#accounts-keys-and-hbar">#accounts-keys-and-hbar</a></td></tr><tr><td align="center"><mark style="color:purple;"><strong>DEVELOPER TOOLS</strong></mark></td><td></td><td></td><td><a href="./#developer-tools">#developer-tools</a></td></tr></tbody></table>
```

<mark style="color:purple;">The Basics</mark>

```
<table data-view="cards"><thead><tr><th></th><th data-hidden></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td><strong>How to Start Developing on Hedera</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/how-to-develop-on-hedera-back-to-the-basics">https://hedera.com/blog/how-to-develop-on-hedera-back-to-the-basics</a></td></tr><tr><td><strong>How to Look Up Transaction History on Hedera Using Mirror Nodes</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/how-to-look-up-transaction-history-on-hedera-using-mirror-nodes-back-to-the-basics">https://hedera.com/blog/how-to-look-up-transaction-history-on-hedera-using-mirror-nodes-back-to-the-basics</a></td></tr><tr><td><strong>Create Testnet Accounts with ED25519 and ECDSA Keys from the Hedera Portal</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/create-accounts-with-ed25519-and-ecdsa-keys-from-the-hedera-portal">https://hedera.com/blog/create-accounts-with-ed25519-and-ecdsa-keys-from-the-hedera-portal</a></td></tr><tr><td><strong>Staking on Hedera for
```

Developers</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/staking-on-hedera-for-developers-back-to-the-basics</td></tr></tbody></table>

<mark style="color:purple;">Smart Contracts</mark>

Get Started

<table data-view="cards"><thead><tr><th></th><th data-hidden></th><th data-hidden></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td>How to Deploy Smart Contracts on Hedera - Part 1</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-deploy-smart-contracts-on-hedera-part-1-a-simple-getter-and-setter-contract</td></tr><tr><td>How to Deploy Smart Contracts on Hedera - Part 2</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-deploy-smart-contracts-on-hedera-part-2-a-contract-with-hedera-token-service-integration</td></tr></tbody></table>

Send and Receive HBAR

<table data-view="cards"><thead><tr><th></th><th data-hidden></th><th data-hidden></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td>How to Send and Receive HBAR Using Smart Contracts - Part 1: Using the SDK</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-send-and-receive-hbar-using-smart-contracts-part-1-using-the-sdk</td></tr></tbody></table>

Working with Smart Contracts

<table data-view="cards"><thead><tr><th></th><th data-hidden></th><th data-hidden></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td>How to Use Ethers.js to Deploy Smart Contracts on Hedera</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-use-hethers-js-to-deploy-smart-contracts-on-hedera</td></tr><tr><td>How to Get Event Information from Hedera Smart Contracts</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-get-event-information-from-hedera-smart-contracts</td></tr><tr><td>The CREATE2 Opcode on Hedera: An Introduction</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/the-create2-opcode-on-hedera-an-introduction</td></tr><tr><td>How to Pass Zero Token Values to Hedera Contracts</td><td></td><td></td><td>https://hedera.com/blog/how-to-pass-zero-token-values-to-hedera-contracts</td></tr></tbody></table>

href="../../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/how-to-pass-zero-token-values-to-hedera-contracts</td></tr><tr><td>Smart Contract Rent on Hedera - Part 1: What You Need to Know</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/smart-contract-rent-on-hedera-is-coming-what-you-need-to-know</td></tr><tr><td>Smart Contract Rent on Hedera - Part 2: How to Pay</td><td></td><td></td><td>IMG9444 (1) (1).jpg</td><td>https://hedera.com/blog/smart-contract-rent-on-hedera-part-2-how-to-pay</td></tr></tbody></table>

<mark style="color:purple;">NFTs and Fungible Tokens on Hedera</mark>

Get Started with the Hedera Token Service (HTS)

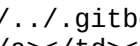
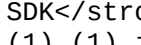
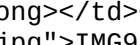
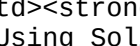
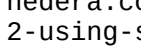
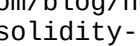
Hedera Token Service - Part 1: How to Mint NFTs			<a >https:="" >img9444="" (1)="" (1).jpg<="" -="" 2:="" 3:="" a><="" a><a="" and="" blog="" delete="" freeze,="" get-started-with-the-hedera-token-service-part-1-how-to-mint-nfts<="" get-started-with-the-hedera-token-service-part-2-kyc-update-and-scheduled-transactions<="" get-started-with-the-hedera-token-service-part-3-how-to-pause-freeze-wipe-and-delete-nfts<="" hedera.com="" how="" href="https://hedera.com/blog/get-started-with-the-hedera-token-service-part-3-how-to-pause-freeze-wipe-and-delete-nfts" kyc,="" nfts<="" p="" part="" pause,="" scheduled="" service="" strong><="" table><="" tbody><="" td><="" td><td><="" td><td><a="" to="" token="" tr><="" tr><tr><td>hedera="" transactions<="" update,="" wipe,="">

Working with HTS Tokens

Mapping Hedera Token Service Standards to ERC20, ERC721, &#x26;			<a >https:="" >img9444="" (1)="" (1).jpg<="" -="" 1:="" 2:="" a><="" a><a="" and="" blog="" contract="" create="" fungible="" hedera="" hedera.com="" how-to-create-hedera-tokens-part-1-fungible-tokens<="" how-to-create-hedera-tokens-part-2-non-fungible-tokens<="" href="https://hedera.com/blog/how-to-create-hedera-tokens-part-2-non-fungible-tokens" mapping-hedera-token-service-standards-to-erc20-erc721-erc1155<="" non-fungible="" p="" part="" sdks="" smart="" strong><="" table><="" tbody><="" td><="" td><td><="" td><td><a="" to="" tokens="" tokens<="" tr><="" tr><tr><td>how="" using="">

  https://hedera.com/blog/how-to-create-hedera-tokens-part-2-non-fungible-tokens	https://hedera.com/blog/how-to-create-hedera-tokens-part-2-non-fungible-tokens
How to Manage Hedera Tokens using Smart Contracts	
  https://hedera.com/blog/how-to-manage-hedera-tokens-using-smart-contracts	https://hedera.com/blog/how-to-manage-hedera-tokens-using-smart-contracts
How to Send and Receive Hedera Tokens Using Smart Contracts - Part 1: SDKs	
  https://hedera.com/blog/how-to-send-and-receive-hedera-tokens-using-smart-contracts-part-1-sdks	https://hedera.com/blog/how-to-send-and-receive-hedera-tokens-using-smart-contracts-part-1-sdks
How to Send and Receive Hedera Tokens Using Smart Contracts - Part 2: Solidity	
  https://hedera.com/blog/how-to-send-and-receive-hedera-tokens-using-smart-contracts-part-2-solidity	https://hedera.com/blog/how-to-send-and-receive-hedera-tokens-using-smart-contracts-part-2-solidity
How to Exempt Hedera Accounts from Custom Token Fees	
  https://hedera.com/blog/how-to-exempt-hedera-accounts-from-custom-token-fees	https://hedera.com/blog/how-to-exempt-hedera-accounts-from-custom-token-fees
NFT Royalty Fees: Everything You Need To Know (Edge-cases Included)	
  https://hedera.com/blog/nft-royalty-fees-hedera-hashgraph	https://hedera.com/blog/nft-royalty-fees-hedera-hashgraph

Accounts, Keys, and HBAR

How to Approve HBAR Allowances on Hedera Using the SDK			
  https://hedera.com/blog/how-to-approve-hbar-allowances-on-hedera-using-the-sdk	https://hedera.com/blog/how-to-approve-hbar-allowances-on-hedera-using-the-sdk		
How to Approve Fungible Token and NFT Allowances on Hedera - Part 1: Using the SDK			
  https://hedera.com/blog/how-to-approve-fungible-token-and-nft-allowances-on-hedera-part-1-using-the-sdk	https://hedera.com/blog/how-to-approve-fungible-token-and-nft-allowances-on-hedera-part-1-using-the-sdk		
How to Approve Fungible Token and NFT Allowances on Hedera - Part 2: Using Solidity Libraries			
  https://hedera.com/blog/how-to-approve-fungible-token-and-nft-allowances-on-hedera-part-2-using-solidity-libraries	https://hedera.com/blog/how-to-approve-fungible-token-and-nft-allowances-on-hedera-part-2-using-solidity-libraries		
How to Approve Fungible Token and NFT Allowances on Hedera - Part 3: ERC Standard Calls			
  https://hedera.com/blog/how-to-approve-allowances-on-hedera-part-3-erc-standard-calls	https://hedera.com/blog/how-to-approve-allowances-on-hedera-part-3-erc-standard-calls		

<mark style="color:purple;">Developer Tools</mark>

```
<table data-view="cards"><thead><tr><th></th><th data-hidden></th><th data-hidden></th><th data-hidden data-card-cover data-type="files"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td><strong>How to Set Up Your Own Hedera Local Network Using Docker</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/how-to-set-up-your-own-hedera-local-network-using-docker">https://hedera.com/blog/how-to-set-up-your-own-hedera-local-network-using-docker</a></td></tr><tr><td><strong>Meet Venin, a concise yet powerful SDK alternative for JS Devs</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/meet-venin-a-concise-yet-powerful-sdk-alternative-for-js-devs">https://hedera.com/blog/meet-venin-a-concise-yet-powerful-sdk-alternative-for-js-devs</a></td></tr><tr><td><strong>Now Available: Hedera JavaScript DID SDK</strong></td><td></td><td></td><td><a href="../../gitbook/assets/IMG9444 (1) (1).jpg">IMG9444 (1) (1).jpg</a></td><td><a href="https://hedera.com/blog/now-available-hedera-javascript-did-sdk">https://hedera.com/blog/now-available-hedera-javascript-did-sdk</a></td></tr></tbody></table>
```

schedule-your-first-transaction.md:

Schedule Your First Transaction

Summary

In this tutorial, you'll learn how to create and sign a scheduled transaction. Scheduled Transactions enable multiple parties to easily, inexpensively, and natively schedule and execute any type of Hedera transaction together. Once a transaction is scheduled, additional signatures can be submitted via a ScheduleSign transaction. After the last signature is received within the allotted timeframe, the scheduled transaction will execute.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

Table of Contents

1. Create Transaction
2. Schedule Transaction
3. Submit Signatures
4. Verify Schedule was Triggered
5. Verify Scheduled Transaction Executed

1. Create a transaction to schedule

First, you will need to build the transaction to schedule. In the example below, you will create a transfer transaction. The sender account has a threshold key

structure that requires 2 out of the 3 keys to sign the transaction to authorize the transfer amount.

```
{% tabs %}
{% tab title="Java" %}
java
//Create a transaction to schedule
TransferTransaction transaction = new TransferTransaction()
    .addHbarTransfer(senderAccount, Hbar.fromTinybars(-1))
    .addHbarTransfer(recipientAccount, Hbar.fromTinybars(1));

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create a transaction to schedule
const transaction = new TransferTransaction()
    .addHbarTransfer(senderAccount, Hbar.fromTinybars(-1))
    .addHbarTransfer(recipientAccount, Hbar.fromTinybars(1));

{% endtab %}

{% tab title="Go" %}
go
//Create a transaction to schedule
transaction := hedera.NewTransferTransaction().
    AddHbarTransfer(senderAccount, hedera.HbarFromTinybar(-1)).
    AddHbarTransfer(recipientAccount, hedera.HbarFromTinybar(1))

{% endtab %}
{% endtabs %}
```

2. Schedule the transfer transaction

Next, you will schedule the transfer transaction by submitting a `ScheduleCreate` transaction to the network. Once the transfer transaction is scheduled, you can obtain the schedule ID from the receipt of the `ScheduleCreate` transaction. The schedule ID identifies the schedule that scheduled the transfer transaction. The schedule ID can be shared with the three signatories. The schedule is immutable unless the admin key is specified during creation.

The scheduled transaction ID of the transfer transaction can also be returned from the receipt of the `ScheduleCreate` transaction. You will notice that the transaction ID for a scheduled transaction includes a `?scheduled` flag e.g. `0.0.9401@1620177544.531971543?scheduled`. All transactions that have been scheduled will include this flag.

You can optionally add signatures you may have during the creation of the `ScheduleCreate` transaction by calling `.freezeWith(client)` and `.sign()` methods. This might make sense if you are one of the required signatures for the scheduled transaction.

Visit the page below to view additional properties that can be set when building a `ScheduleCreate` transaction.

```
{% content-ref url="../../../sdks-and-apis/sdks/schedule-transaction/create-a-schedule-transaction.md" %}
create-a-schedule-transaction.md
{% endcontent-ref %}

{% tabs %}
```

```

{% tab title="Java" %}
java
//Schedule a transaction
TransactionResponse scheduleTransaction = new ScheduleCreateTransaction()
    .setScheduledTransaction(transaction)
    .execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt = scheduleTransaction.getReceipt(client);

//Get the schedule ID
ScheduleId scheduleId = receipt.scheduleId;
System.out.println("The schedule ID is " +scheduleId);

//Get the scheduled transaction ID
TransactionId scheduledTxId = receipt.scheduledTransactionId;
System.out.println("The scheduled transaction ID is " +scheduledTxId);

{% endtab %}

{% tab title="JavaScript" %}
<pre class="language-javascript"><code class="lang-javascript">
//Schedule a transaction
<strong>const scheduleTransaction = await new ScheduleCreateTransaction()
</strong>    .setScheduledTransaction(transaction)
    .execute(client);

//Get the receipt of the transaction
const receipt = await scheduleTransaction.getReceipt(client);

//Get the schedule ID
const scheduleId = receipt.scheduleId;
console.log("The schedule ID is " +scheduleId);

//Get the scheduled transaction ID
const scheduledTxId = receipt.scheduledTransactionId;
console.log("The scheduled transaction ID is " +scheduledTxId);
</code></pre>
{% endtab %}

{% tab title="Go" %}
go
//Schedule a transaction
scheduleTransaction, err := hedera.NewScheduleCreateTransaction().
    SetScheduledTransaction(transaction)
if err != nil {
    panic(err)
}

submitScheduleTx, err := scheduleTransaction.Execute(client)
if err != nil {
    panic(err)
}

//Get the receipt of the transaction
receipt, err := submitScheduleTx.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the schedule ID
scheduleId := receipt.ScheduleID
fmt.Printf("The schedule ID %v\n", scheduleId)

```

```
//Get the scheduled transaction ID
scheduleTxId := receipt1.ScheduledTransactionID
fmt.Printf("The scheduled transaction ID is %v\n", scheduleTxId)

{% endtab %}
{% endtabs %}
```

3. Submit Signatures

Submit one of the required signatures for the transfer transaction

The signatures are submitted to the network via a ScheduleSign transaction. The ScheduleSign transaction requires the schedule ID of the schedule and the signature of one or more of the required keys. The scheduled transaction has 30 minutes from the time it is scheduled to receive all of its signatures; if the signature requirements are not met, the scheduled transaction will expire.

In the example below, you will submit one signature, confirm the transaction was successful, and get the schedule info to verify the signature was added to the schedule. To verify the signature was added, you can compare the public key of the submitted signature to the public key that is returned from the schedule info request.

```
{% tabs %}
{% tab title="Java" %}
java
//Submit the first signatures
TransactionResponse signature1 = new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey1)
    .execute(client);

//Verify the transaction was successful and submit a schedule info request
TransactionReceipt receipt1 = signature1.getReceipt(client);
System.out.println("The transaction status is " +receipt1.status);

ScheduleInfo query1 = new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);

//Confirm the signature was added to the schedule
System.out.println(query1);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Submit the first signature
const signature1 = await (await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey1))
    .execute(client);

//Verify the transaction was successful and submit a schedule info request
const receipt1 = await signature1.getReceipt(client);
console.log("The transaction status is " +receipt1.status.toString());

const query1 = await new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);
```

```

//Confirm the signature was added to the schedule
console.log(query1);

{% endtab %}

{% tab title="Go" %}
go
//Submit the first signature
signature1, err := hedera.NewScheduleSignTransaction().
    SetScheduleID(scheduleId).
    FreezeWith(client)
if err != nil {
    panic(err)
}

//Verify the transaction was successful and submit a schedule info request
submitTx, err := signature1.Sign(signerKey1).Execute(client)
if err != nil {
    panic(err)
}

receipt1, err := submitTx.GetReceipt(client)
if err != nil {
    panic(err)
}

status := receipt1.Status

fmt.Printf("The transaction status is %v\n", status)

query1, err := hedera.NewScheduleInfoQuery().
    SetScheduleID(scheduleId).
    Execute(client)

//Confirm the signature was added to the schedule
fmt.Print(query1)

```

```

{% endtab %}
{% endtabs %}

```

Submit the second signature

Next, you will submit the second signature and verify the transaction was successful by requesting the receipt. For example purposes, you have access to all three signing keys. But the idea here is that each signer can independently submit their signature to the network.

```

{% tabs %}
{% tab title="Java" %}
{% code title="Java" %}
java
//Submit the second signature
TransactionResponse signature2 = new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey2)
    .execute(client);

//Verify the transaction was successful
TransactionReceipt receipt2 = signature2.getReceipt(client);
System.out.println("The transaction status" +receipt2.status);

{% endcode %}

```

```

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Submit the second signature
const signature2 = await (await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(signerKey2))
    .execute(client);

//Verify the transaction was successful
const receipt2 = await signature2.getReceipt(client);
console.log("The transaction status " +receipt2.status.toString());

{% endtab %}

{% tab title="Go" %}
go
//Submit the second signature
signature2, err := hedera.NewScheduleSignTransaction().
    SetScheduleID(scheduleId).
    FreezeWith(client)
if err != nil {
    panic(err)
}

//Verify the transaction was successful and submit a schedule info request
submitTx2, err := signature2.Sign(signerKey2).Execute(client)
if err != nil {
    panic(err)
}

receipt2, err := submitTx2.GetReceipt(client)
if err != nil {
    panic(err)
}

status2 := receipt2.Status

fmt.Printf("The transaction status is %v\n", status2)

{% endtab %}
{% endtabs %}

```

4. Verify the schedule was triggered

The schedule is triggered after it meets its minimum signing requirements. As soon as the last required signature is submitted, the schedule executes the scheduled transaction. To verify the schedule was triggered, query for the schedule info. When the schedule info is returned, you should notice both public keys that signed in the signatories field and the timestamp recorded for when the schedule transaction was executed in the executedAt field.

```

{% tabs %}
{% tab title="Java" %}
java
//Get the schedule info
ScheduleInfo query2 = new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);

```

```

System.out.println(query2);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Get the schedule info
const query2 = await new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);

console.log(query2);

{% endtab %}

{% tab title="Go" %}
go
//Get the schedule info
query2, err := hedera.NewScheduleInfoQuery().
    SetScheduleID(scheduleId).
    Execute(client)

fmt.Print(query2)

{% endtab %}
{% endtabs %}

```

5. Verify the scheduled transaction executed

When the scheduled transaction (transfer transaction) executes a record is produced that contains the transaction details. The scheduled transaction record can be requested immediately after the transaction has executed and includes the corresponding schedule ID. If you do not know when the scheduled transaction will execute, you can always query a mirror node using the scheduled transaction ID without the ?scheduled flag to get a copy of the transaction record.

```

{% tabs %}
{% tab title="Java" %}
java
//Get the scheduled transaction record
TransactionRecord scheduledTxRecord =
TransactionId.fromString(scheduledTxId.toString()).getRecord(client);
System.out.println("The scheduled transaction record is: " +scheduledTxRecord);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Get the scheduled transaction record
const scheduledTxRecord = await
TransactionId.fromString(scheduledTxId.toString()).getRecord(client);
console.log("The scheduled transaction record is: " +scheduledTxRecord);

{% endtab %}

{% tab title="Go" %}
go
scheduledTxRecord, err := scheduledTransactionId.GetRecord(client)
fmt.Printf("The scheduled transaction record is %v\n", scheduledTxRecord)

{% endtab %}
{% endtabs %}

```

```
{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}
```

README.md:

Get Started with the Hedera Consensus Service Fabric Plugin

```
{% hint style="warning" %}
You must have a basic understanding of the Hyperledger Fabric network, its key
concepts, first-network sample, and transaction flow. Please visit the
Hyperledger Fabric docs to familiarize yourself with these concepts if you have
not done so already.
{% endhint %}
```

Background

Hyperledger Fabric is one of the most popular private/permissioned enterprise blockchain frameworks available today. Fabric's modular architecture approach enables users to specify network components like network members and choice of consensus protocol (ordering service).

In this tutorial, you will create a Hyperledger Fabric network that leverages the Hedera Consensus Service Fabric plug-in to use Hedera as the ordering service via the first-network sample. The HCS Hyperledger Fabric network will be composed of two organizations. Each organization will host two peer nodes. The two organizations will privately communicate and transact within a Hyperledger Fabric channel.

If you already have experience with the Hyperledger Fabric network and the first-network sample, you can skip down to the requirements section to get started.

Transaction Flow

1. A client application creates a transaction proposal and sends it to Hyperledger Fabric network peers. The transaction proposal is endorsed (signed) by the Hyperledger Fabric network peers. The endorsed transaction proposal is sent back to the client application.
2. Client application submits the endorsed transaction to the Hyperledger Fabric network HCS ordering node. That Hyperledger Fabric ordering node interacts with the HCS fabric plugin as the ordering service
3. Fabric transactions are fragmented into many messages and submitted to the Hedera network against a particular topic ID
4. Hedera mainnet nodes timestamp and order the fragmented messages
5. Hedera mirror nodes read the fragmented messages in consensus order from the Hedera mainnet
6. Hedera mirror node relays the ordered messages back to the Hyperledger Fabric ordering node
7. Hyperledger Fabric ordering nodes reassembles the fragmented messages into fabric transactions, form fabric blocks, and communicate the blocks to the Hyperledger Fabric peer nodes
8. Hyperledger Fabric peer nodes update to have the same world state

```

```

Concepts

A basic description of the concepts for the purposes of this tutorial.

Client Application

A client application communicates to a Hyperledger Fabric peer node to submit a transaction proposal and collects the endorsed (signed) transaction proposal. The transaction proposal may initiate a modification of the current state of the distributed ledger.

Hedera Consensus Service Fabric Plugin

A consensus service plugin for the Hyperledger Fabric network that leverages the Hedera Consensus Service to order transactions using the unique properties of the hashgraph consensus algorithm that powers the Hedera network.

Hedera Mirror Node

A Hedera mirror node reads the ordered messages from a Hedera mainnet node and communicates that information back to the Hyperledger Fabric network. Each message has the following information:

- ConsensusTimestamp
- Message Body
- Topic Running Hash
- Topic Sequence Number

Hedera Mainnet Node

A Hedera mainnet node receives the Fabric transaction in fragmented message transactions from the HCS Hyperledger ordering node. That mainnet node gossips the transactions to the other mainnet nodes which collectively assigns the transactions a consensus timestamp and order within the corresponding topic.

fabric-hcs Repository

The project that contains all the necessary files to run the HCS Hyperledger Fabric network sample.

Hyperledger Fabric Network

The Hyperledger Fabric network is composed of peer nodes, ordering nodes, chaincode (smart contracts), organizations, channels, a distributed ledger and an ordering service.

Hyperledger Fabric Network Peers

Peers are nodes in the Hyperledger Fabric network that host instances of the ledger and instances of the chaincode. Client applications interact with the peer nodes to endorse a transaction proposal.

Hyperledger Fabric Channels

Channels are a mechanism by which organizations can communicate and transact privately with one another.

Hyperledger Fabric Organizations

Organizations are participants that would like to communicate and transact privately with one another.

Hyperledger Fabric Orderer

A node that orders the transactions received from a client application (also known as an "ordering node"). In this case, the ordering node interacts with the HCS fabric plugin as the ordering service.

Requirements:

```
{% hint style="info" %}
```

If you would like to run the sample using a virtual environment, please follow the instructions here.

```
{% endhint %}
```

Hedera Consensus Service

Testnet account ID and private key

Please follow the instructions here

pluggable-hcs Repository

You will be directed to clone this repository in the outlined steps below

```
{% hint style="info" %}
```

The HCS Fabric plugin supports Fabric ordering service 2.0 and is compatible with older versions peers.

```
{% endhint %}
```

Hyperledger Fabric Network

Download and install the following if you do not already have them on your computer.

Git

Download Git

Check to see if you have it installed from your terminal: `git --version`

cURL

Download cURL

Check to see if you have it installed from your terminal: `curl --version`

wget

Install from your terminal (MacOS): `brew install wget`

Check to see if you have it installed from your terminal: `wget --version`

Docker and Docker Compose

Download Docker

Docker version 17.06.2-ce or greater is required

Check to see if you have it installed from your terminal: `docker --version` &&

`docker-compose --version`

Go Programming Language

Download Go

Go version 1.13.x is required

Check to see if you have it installed from your terminal: `go version`

make

Install from your terminal (MacOS): `brew install make`

Check to see if you have installed from your terminal: `make --version`

gcc

Check to see if you have it installed from your terminal: `gcc --version`

Additionally, you may reference Hyperledger Fabric's documentation for the prerequisites.

Terminal/IDE

You should be able to use the commands provided in this tutorial in terminal prompt or IDE of choice.

1. Clone the pluggable-hcs Project

Open your terminal or favorite IDE

Enter the following commands to set-up your environment variables (required)

Optionally you can make these settings permanent by placing them in the appropriate startup file, such as your personal `/.bashrc` file if you are using the bash shell under Linux.

```
$ export GOPATH=$HOME/go
$ export PATH=$PATH:$GOPATH/bin
```

Create a hyperledger directory and navigate into that directory

```
$ mkdir -p $GOPATH/src/github.com/hyperledger && cd
$GOPATH/src/github.com/hyperledger
```

Clone the pluggable-hcs repository and rename it to fabric
You must rename the folder to fabric otherwise you will run into issues in the following steps

```
$ git clone https://github.com/hyperledger-labs/pluggable-hcs fabric
$ cd fabric
```

Confirm you are on the main branch

```
$ git branch
```

☆ You have now successfully set your Go path variables and installed the pluggable-hcs/fabric repository.

2. Build Fabric Binaries and Docker Images

cd to the fabric directory if you are not there already from your terminal or favorite IDE

```
$ cd fabric
```

Follow the commands below to build the required fabric binaries and docker images

Note: This process may take a few minutes to complete

Note: Newer versions of Docker enable gRPC FUSE file sharing implementation by default which result in a failure. Switch to osxfs (Legacy) in Docker Settings

```
$ make clean
$ make configtxgen configtxlator cryptogen orderer peer docker
```

3. Hedera Network & HCS Hyperledger Fabric Orderer Configuration

You will now enter your Hedera testnet account ID and private key information to the relevant configuration files. If you have not previously generated your testnet account, please follow the instructions here.

Navigate to the first-network directory from your terminal or favorite IDE

```
$ cd first-network
```

Edit the hedera\env.json file via the terminal or IDE

This sets up the Hedera environment and allows you to create topics and submit messages to the Hedera network. The Hedera account entered here pays for the transaction fees associated with creating and submitting messages.

If you are using an IDE, you may skip this command and edit the file

```
$ nano hederaenv.json
```

Enter your Hedera account ID (e.g. 0.0.1234) in the operatorId field
Enter your account private key in the operatorKey field

```
javascript
{
  "operatorId": "put your testnet account id here",
  "operatorKey": "put your account private key here",
  "nodeId": "",
  "nodeAddress": "",
  "network": {
    "0.testnet.hedera.com:50211": "0.0.3",
    "1.testnet.hedera.com:50211": "0.0.4",
    "2.testnet.hedera.com:50211": "0.0.5",
    "3.testnet.hedera.com:50211": "0.0.6"
  },
  "mirrorNodeAddress": "hcs.testnet.mirrornode.hedera.com:5600"
}
```

Save the hedera\env.json file

Open the orderer.yaml file from your terminal or favorite IDE

Here you will set the configurations for the HCS Hyperledger Fabric orderer node

This file configures hcs as the ordering service

This also contains the node address and node ID to interact with the Hedera mainnet nodes

The Hedera mirror node address to receive ordered transaction from

If you are using an IDE, you may skip this command and edit the file

```
$ nano orderer.yaml
```

Scroll down to the "SECTION: Hcs" heading

Enter your Hedera account ID in the Operator.Id field

Enter your Hedera account private key in the Operator.PrivateKey.Key field

```
yaml
operator set for the orderer
Operator:
  Id: Your Hedera testnet account id here
  PrivateKey:
    type of the private key
    Type: ed25519
    key string (hex or PEM encoded) or path to the key file
    Key: Your Hedera testnet account private key string here
```

Save the orderer.yaml file

```
{% hint style="info" %}
```

Please make sure you have entered your Hedera information correctly with no syntax errors as it will cause issues when trying to run the network later.

```
{% endhint %}
```

☆ You have now successfully set up your configuration variables for the Hedera operator and for the HCS Hyperledger ordering node.

4. Run Your Network

In this step you will create your HCS Hyperledger Fabric Network.

Make sure you are within the first-network directory before running these commands.

Enter the following command to start the HCS Hyperledger Fabric network

```
$ ./byfn.sh up -t 20
```

Enter "Y" to accept the 20 second timeout and CLI delay of 3 seconds

```
{% hint style="info" %}
```

Please note it's necessary to set the timeout to 20 seconds since otherwise some checks in the script may fail due to HCS having a larger consensus delay than raft/kafka based ordering service.

```
{% endhint %}
```

The script generates two HCS topics

One topic will be for the Hyperledger Fabric system channel

One topic will be for the Hyperledger Fabric application channel

HCS topics in this example are configured so that anyone can submit messages to them

HCS messages can be configured to be private topics where the sender would require the submitKey to successfully publish messages to them

Transactions submitted to either of these channels will be visible from a mirror node explorer

Transactions can be fragmented into smaller chunks resulting in multiple HCS messages for a single transaction

The sample is almost identical with the upstream first-network sample as of 2.0.0 release. In summary, the modifications are:

1. Updated docker compose files to use hcs-based orderers.
2. Added a function in byfn.sh to generate a 256-bit AES key for data encryption/decryption between fabric orderers and hedera testnet.
3. Added a function in byfn.sh to use the hcscli tool for HCS topic creation and mapping topics to fabric channels.

A successful run will end with the following message:

```
\===== All GOOD, BYFN execution completed =====
```

[illegible]

5. Tear down the network

It is required to run the following command to tear down the network

If you do not tear down the network and try to restart the network you may run into issues

```
$ ./byfn.sh down
```

6. Verify Topics & Messages

Topics and messages created in this tutorial can be verified on any available mirror node explorer

At the start of the script, you can see the two HCS topic IDs that were created

```
installing hcscli ...\
```

```
generated HCS topics: 0.0.23419 0.0.23420\
```

```
0.0.23419 will be used for the system channel, and 0.0.23420 will be used for the application channel
```

Visit a Hedera mirror node explorer to verify the topics and messages that were created on testnet by searching the two topic IDs

You will be able to view the application channel and system channel topics and all associated messages from this example

A single Fabric transaction sent to an ordering node could result in multiple HCS consensus messages as HCS messages have a 6k message size limit

i.e. there may not be a 1:1 correlation between a Fabric transaction and HCS message

A fabric transaction payload is encrypted by the ordering node therefore the subsequent HCS transaction payload is also encrypted

All messages on the mirror node explorer will be displayed in encrypted format.

```
{% hint style="info" %}
```

Make sure you have selected the testnet network toggle in the explorer as the topics and messages created through this tutorial will not appear on the main network.

```
{% endhint %}
```

You have successfully done the following:

Created a Hyperledger Fabric network using the HCS Fabric plugin as the ordering service

Verified the topics and messages created in this example network

Running into issues or have suggestions? Visit the developer advocates in Discord and post your comments to the hedera-consensus-service channel 🗨️ .

```
# virtual-environment-set-up.md:
```

Virtual Environment Setup

Enables developers to run the HCS Hyperledger Fabric sample network using a virtual environment set-up.

Requirements

Hedera testnet account ID and account private key

pluggable-hcs repository

Vagrant

Virtual Box

Terminal/IDE

1. Open your terminal/IDE and CD to where you would like to clone the fabric-hcs project

Clone the pluggable-hcs repository and rename the project folder to fabric

Navigate to the fabric folder

```
git clone https://github.com/hyperledger-labs/pluggable-hcs fabric
cd fabric
```

You should now be in the fabric project folder

2. Confirm you are on the master branch

```
git branch
```

3. Navigate to the vagrant folder and start your virtual machine

```
cd vagrant
vagrant up
vagrant ssh
```

You should now be back in the fabric folder

Now you have your virtual environment ready to go. Please refer to step two: Build Fabric Binaries and Docker Images in the master tutorial to continue.

```
# hashio.md:
```

```
---
```

```
description: >-
```

```
  How to configure a JSON-RPC endpoint that enables communication between
  EVM-compatible developer tools using Hashio
```

```
---
```

Configuring Hashio RPC endpoints

Hashio is a public RPC endpoint hosted by Swirlds Labs. As a public endpoint, it:

- Is free to use
- Does not have any sign-up requirements
- Has significantly restrictive rate limits

While this combination may be considered less reliable, it offers the highest levels of ease of use among RPC endpoints.

To connect to the Hedera networks via Hashio, simply use this URL when initializing the wallet or web3 provider instance:

```
{% tabs %}
{% tab title="Hedera Mainnet" %}
```

```
https://mainnet.hashio.io/api
```

```
{% endtab %}
```

```
{% tab title="Hedera Testnet" %}
```

```
https://testnet.hashio.io/api
```

```
{% endtab %}
```

```
{% tab title="Hedera Previewnet" %}
```

```
https://previewnet.hashio.io/api
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
{% hint style="warning" %}
```

Please note: Hashio is For development and testing purposes only. Production use cases are strongly encouraged to use commercial-grade JSON-RPC relays or host their own instance of the Hedera JSON-RPC Relay.

```
{% endhint %}
```

No further settings or configurations are needed!

```
# hedera-json-rpc-relay.md:
```

```
---
```

```
description: >-
```

How to configure a JSON-RPC endpoint that enables the communication between EVM-compatible developer tools using the Hedera JSON-RPC Relay

```
---
```

Configuring Hedera JSON-RPC Relay endpoints

Hedera JSON-RPC Relay is a server run by you on your own computer - decentralization for the win!

As such, it:

- Is free to use on Hedera Previewnet and Hedera Testnet
- Does not have any sign-up requirements
- Does not have any rate limits
- Requires several additional steps required to set it up, plus developer/command line skills

While this combination may be considered less user-friendly, it offers the highest levels of reliability among RPC endpoints.

This also makes the Hedera JSON-RPC Relay a good alternative for local development and testing; and also a potential option for contributing infrastructure to the Hedera ecosystem.

To connect to Hedera networks via your own instance of Hedera JSON-RPC Relay, use this URL when initializing the wallet/ web3 provider instance:

```
{% tabs %}
```

```
{% tab title="Hedera Mainnet" %}
```

<http://localhost:7546>

```
{% endtab %}
```

```
{% tab title="Hedera Testnet" %}
```

<http://localhost:7546>

```
{% endtab %}
```

```
{% tab title="Hedera Previewnet" %}
```

<http://localhost:7546>

```
{% endtab %}
```

```
{% endtabs %}
```

<details>

<summary>Notes</summary>

(1) The RPC endpoint URL, including the port number 7546, is the same for whichever network you intend to connect to: Hedera Previewnet, Hedera Testnet, and Hedera Mainnet. The selection of network depends upon the configuration file, which we will create in subsequent steps.

(2) The hedera-json-rpc-relay server is designed to be able to be deployed in your own cloud instances. For non-production use cases, a Docker compose file is provided. For production use cases Kubernetes Helm charts are provided. However, both the Docker and Kubernetes options are beyond the scope of this tutorial. This tutorial focuses on simply configuring and running the server directly.

</details>

To get this service running, you will need to do the following pre-requisite steps:

(1) Clone the git project:

```
{% code overflow="wrap" %}
shell
git clone -b main --single-branch https://github.com/hashgraph/hedera-json-rpc-relay.git
```

```
{% endcode %}
```

<details>

<summary>Alternative with <code>git</code> and SSH</summary>

If you have configured SSH to work with git, you may wish use this command instead:

```
{% code overflow="wrap" %}
shell
git clone -b main --single-branch git@github.com:hashgraph/hedera-json-rpc-relay.git
```

```
{% endcode %}
```

</details>

(2) Enter the directory that you have cloned, and install dependencies. It is recommended that you have NodeJS version 20 or later for this.

```
sh
cd hedera-json-rpc-relay
npm install
```

(3) Build the project, including its sub-packages.

```
sh
npm run build
```

(4) Create a file named .env in the root directory of this project by copying .env.example and naming it .env.

```
shell
cp .env.example .env
```

Then set the following fields:

```
{% tabs %}
{% tab title="Hedera Mainnet" %}
{% @github-files/github-code-block url="https://github.com/hashgraph/hedera-
json-rpc-relay/blob/f9d5ebaa80/docs/examples/.env.mainnet.sample" %}
{% endtab %}
```

```
{% tab title="Hedera Testnet" %}
{% @github-files/github-code-block url="https://github.com/hashgraph/hedera-
json-rpc-relay/blob/f9d5ebaa80/docs/examples/.env.testnet.sample" %}
{% endtab %}
```

```
{% tab title="Hedera Previewnet" %}
{% @github-files/github-code-block url="https://github.com/hashgraph/hedera-
json-rpc-relay/blob/f9d5ebaa80/docs/examples/.env.previewnet.sample" %}
{% endtab %}
{% endtabs %}
```

```
{% hint style="success" %}
```

The following steps require that you already have an account on the Hedera network that you are connecting to. This account should be funded with some HBAR.

If you have yet to set one up, for Testnet you may use the Hedera Faucet. See Hedera Faucet instructions.

For either Previewnet or Testnet you may use the Hedera Portal. See Hedera Developer Portal Profile instructions.

Note that setting up a Mainnet account and funding it is out of scope for this article.

```
{% endhint %}
```

(5a) Copy your Account ID value into the .env file in the OPERATORIDMAIN field.

(5b) Copy your account's DER Encoded Private Key into the .env file in the OPERATORKEYMAIN field.

For example, if your account ID is 0.0.12345, your private key is a1b2c3, and you are connecting to Testnet, the .env file should look like the following.

```
sh
HEDERANETWORK=testnet
OPERATORIDMAIN=0.0.12345
OPERATORKEYMAIN=302e0201...
CHAINID=0x128
MIRRORNODEURL=https://testnet.mirrornode.hedera.com/
```

<details>

<summary>"Operator" and "payer account" concepts</summary>

Like other EVM-compatible networks, transactions must be paid for in the native currency. This is true for Hedera as well, where all transactions are paid for, denominated in HBAR.

Unlike other EVM-compatible networks, when an EVM transaction is submitted on a Hedera network, that transaction can be paid for by a different "payer account". The hedera-json-rpc-relay takes care of this automatically for you, wrapping the transaction. This is why there is a need for an OPERATORIDMAIN and OPERATORKEYMAIN, as this is the "payer account".

This effectively means that running an instance of hedera-json-rpc-relay on Hedera Mainnet is not free. On other Hedera networks, e.g. Hedera Testnet, where HBAR are obtained for free, it is effectively free. Apart from HBAR costs, the relay service is indeed free to use, and you are really limited only by your own hardware.

</details>

(6) Run `npm run start` to start the RPC relay server.

Now you have an instance of Hedera JSON-RPC Relay running locally, and you are ready to send RPC requests to your selected Hedera network!

```
{% hint style="success" %}
Full reference configuration options for Hedera JSON-RPC Relay:
docs/configuration.md.
{% endhint %}
```

README.md:

```
---
description: >-
  A variety of EVM-compatible developer tools and wallets communicate over
  JSON-RPC, so here is how to set them up!
---
```

How to Connect to Hedera Networks Over RPC

There are four options to establish a connection to Hedera Networks. Validation Cloud, Arkhia, and Hashio are all managed JSON-RPC providers; the Hedera JSON-RPC Relay is for running your own local JSON-RPC instance. Hashio is a Hedera community service offered by Hashgraph that has limited capabilities.

- ☐ Hashio\\
- ☐ Hedera JSON-RPC Relay
- ☐ Validation Cloud
- ☐ QuickNode (docs)

<p>Writer: Brendan, DevRel Engineer</p> <p>GitHub Blog</p>
<p>Writer: Aaron, Validation Cloud</p> <p>GitHub https://github.com/aaron-cottrell-vc</p>
<p>Editor: Krystal, Technical Writer</p> <p>GitHub Twitter</p> <p>https://twitter.com/theekrystallee</p>

```
{% hint style="warning" %}
\\Please note: Hashio is For development and testing purposes only. Production
use cases are strongly encouraged to use commercial-grade JSON-RPC relays or
host their own instance of the Hedera JSON-RPC Relay.
{% endhint %}
```

validation-cloud.md:

Configuring Validation Cloud RPC endpoints

Validation Cloud is a third-party organization that runs a JSON-RPC and Mirror Node managed service for Hedera as well as other popular blockchain networks. It is a "freemium" offering, meaning it has a free tier and a paid offering. As a managed service, it:

- Is free to use up to a point, with a free usage allowance that resets every month

- Does not require a credit card to sign up, only an email address

- Does not apply specific rate limits and can scale to be used by high volume apps

This combination makes it fairly straightforward to use and more reliable than the public RPC endpoint.

To connect to Hedera networks via Validation Cloud, simply use this URL when initializing the wallet or web3 provider instance:

```
{% tabs %}
```

```
{% tab title="Hedera Mainnet" %}
```

```
https://mainnet.hedera.validationcloud.io/v1/<YOURAPIKEY>
```

```
{% endtab %}
```

```
{% tab title="Hedera Testnet" %}
```

```
https://testnet.hedera.validationcloud.io/v1/<YOURAPIKEY>
```

```
{% endtab %}
```

```
{% endtabs %}
```

```
{% hint style="info" %}
```

```
Note: Validation Cloud provides RPC endpoints for Hedera Mainnet and Hedera Testnet but not for Hedera Previewnet.&#x20;
```

```
{% endhint %}
```

You will need to replace <YOURAPIKEY> with a Validation Cloud API Endpoint Key, and that requires the following prerequisite steps:

- (1) Sign up for an account at app.validationcloud.io/api/auth/signup
- (2) Accept the terms of use and then verify your email address
- (3) Click the "Create endpoint" button at the top of the Validation Cloud Node API dashboard:

```
<figure><figcaption></figcaption></figure>
```

- (4) Fill in a name for the endpoint, select Hedera and pick whether you want Testnet or Mainnet, then click on the "Confirm" button:

```
<figure><figcaption></figcaption></figure>
```

- (5) Your key should now be created and you can click the copy to clipboard icon to use it to make requests:

```
<figure><figcaption></figcaption></figure>
```

Now you're ready to connect to an RPC endpoint or query a Mirror Node via Validation Cloud!

You can also watch this short video showing the process end-to-end, with examples of making Hedera JSON-RPC and Mirror Node requests with Validation Cloud:

```
{% embed url="https://www.loom.com/share/22cb87ee589248e58c95bbba6edc1667?sid=a93b0c6d-58a9-40de-a230-509b4d2198a3" %}
```

create-an-hbar-faucet-app-using-react-and-metamask.md:

Create an HBAR Faucet App Using React and MetaMask

Enabling the opportunity to connect to a decentralized application (dApp) with different wallets provides more control to the user. Recently, HIP-583 added the necessary network infrastructure to support Ethereum Virtual Machine (EVM) wallets on the Hedera network. This added functionality, combined with the auto-create flow described in HIP-32, enables developers to transfer native Hedera Token Service (HTS) tokens to EVM addresses that do not yet exist on the Hedera network. In this tutorial, we start out with a Hedera react app, connect our dApp to MetaMask, and finally transfer HBAR to the connected MetaMask account.

What we will do:

This tutorial will show you how to create a Hedera React app using TypeScript and Material UI. You'll install the MetaMask Chrome extension, add the Hedera Testnet network, connect your dApp to it, and even send some HBAR to your MetaMask wallet.

<details>

<summary>Project Repos</summary>

The complete TypeScript project can be found on GitHub [here](#).

The complete JavaScript project can be found on GitHub [here](#).

</details>

Prerequisites

- Create a Hedera Testnet account [here](#).
- Install the MetaMask Chrome extension. [#x20;](#)
- Basic understanding of TypeScript and React.
- Set up your environment and create your client [here](#). [#x20;](#)

Table of Contents

1. Create React App
2. Configure MetaMask
3. Connect MetaMask
4. Install Dependencies
5. Create Client
6. Send HBAR to MetaMask
7. Summary
8. Additional Resources

Create React App

Open your terminal and run the following command to create a React app that utilizes TypeScript and Material UI.

```
bash
npx create-react-app <app-name> --template hedera-theme
```

This creates react-app theme that provides a navbar with a button, footer, react-router, and a global context for state management. We will use this template to kickstart our project.

Open the project in Visual Studio code or your IDE of choice. Your project directory structure will look like this:

```
|— nodemodules
|— publicLA
|— src
|— .gitignore
|— package-lock.json
|— package.json
|— README.md
|— tsconfig.json
```

If you have not already installed the MetaMask Chrome extension, please install it [here](#).

Configure MetaMask with Hedera Testnet Network

In this step, we will add the code necessary to add the Hedera Testnet network to MetaMask so we can connect to it. Before we do that, let's check out our project folder structure, which looks like this:

```
src
|— assets
|— components
|— App.tsx
|— AppRouter.tsx
|— Home.tsx
|— index.tsx
|— react-app-env.ts
```

Let's continue and add a new folder inside src and name it services. Inside the services folder, add a file named metamaskService.ts. Your src folder structure will look like this:

```
src
|— assets
|— components
|— services
|   |— metamaskServices.ts
|— App.tsx
|— AppRouter.tsx
|— Home.tsx
|— index.tsx
```

└─ react-app-env.ts

We will store the code in the services folder, which will be used to support certain services required to run our application. We added the `metamaskService.ts` file to hold the code necessary for our MetaMask integration with our application. This distinction helps keep our code organized and clean.

It's important to note that to add the Hedera test network to MetaMask, we must determine if MetaMask exists in our browser. We can achieve this by using the window, aka the Browser Object Model (BOM), which represents the browser's window. This window object can access all global JavaScript objects, functions, and variables. If we are successful in accessing the Ethereum object off of the window object, then it is an indication that MetaMask is installed. If it is not successful, then we will handle that by throwing an error message to the console that reads, "MetaMask is not installed! Go install the extension!"

Let's begin!

In our `metamaskService.ts` file, create a function named `switchToHederaNetwork`.

```
typescript
export const switchToHederaNetwork = async (ethereum: any) => {
  try {
    await ethereum.request({
      method: 'walletswitchEthereumChain',
      params: [{ chainId: '0x128' }] // chainId must be in hexadecimal numbers
    });
  } catch (error: any) {
    if (error.code === 4902) {
      try {
        await ethereum.request({
          method: 'walletaddEthereumChain',
          params: [
            {
              chainName: 'Hedera Testnet',
              chainId: '0x128',
              nativeCurrency: {
                name: 'HBAR',
                symbol: 'HBAR',
                decimals: 18
              },
              rpcUrls: ['https://testnet.hashio.io/api']
            }
          ],
        });
      } catch (addError) {
        console.error(addError);
      }
    }
    console.error(error);
  }
}
```

Let's digest this function a little further. This function starts by submitting a request to change to the network to Hedera Testnet. If it fails to connect due to the Hedera Testnet network not being configured in MetaMask, then we will submit a request to add a new chain.\

In the request to add a new chain, the decimal value is set to 18 even though HBAR has 8 decimals. The reason for this is that MetaMask only supports chains that have 18 decimals. The RPC URL we use is `https://testnet.hashio.io/api`, which comes from Hashio, the SwirlsLabs-hosted version of the JSON-RPC Relay.

Connect Our dApp to MetaMask and Retrieve Wallet Address

We have added the code necessary to configure MetaMask with the Hedera test network. Now, it's time to focus on adding the code that will allow us to connect our dApp to MetaMask.

In the `metamaskService.ts` file underneath `switchToHederaNetwork()`, add a new function named `connectToMetamask()`.

```
typescript
// returns a list of accounts
// otherwise empty array
export const connectToMetamask = async () => {
  const { ethereum } = window as any;
  // keep track of accounts returned
  let accounts: string[] = []
  if (!ethereum) {
    throw new Error("Metamask is not installed! Go install the extension!");
  }

  switchToHederaNetwork(ethereum);

  accounts = await ethereum.request({
    method: 'ethrequestAccounts',
  });
  return accounts;
}
```

If MetaMask is installed, `connectToMetamask()` will call `switchToHederaNetwork()` and submit a request to get accounts.

Before we do any kind of work or testing, it is important to ensure we are connected to the correct network.

Install Dependencies

We've written the code to connect our application to MetaMask. Now it's time to send HBAR to our MetaMask wallet.

Install the Hedera JavaScript SDK and `dotenv` by running the following command in the project root directory:

```
bash
npm install --save @hashgraph/sdk dotenv
```

Create your `.env` file by adding a new file to the root directory of your `react-app` and naming it `.env`. Next, add the `.env` file to your `.gitignore` to prevent sharing your credentials in source control.

Your `.env` file will look like this:

```
REACTAPPMYACCOUNTID=
REACTAPPMYPRIVATEKEY=
```

In a `react-app`, the environment variables in your `.env` file must start with

REACT\APP\. Set the values to the testnet account you created through the Hedera developer portal.

Note: If you need to create a Hedera Testnet account, visit portal.hedera.com and register to receive 10,000 testnet HBAR.

Create Your Client

A client is used to communicate with the network. We create our client for the Hedera Testnet, which enables us to submit transactions and pay for them. Let's create our client in our Home.tsx file.

```
typescript
export default function Home() {
  // If we weren't able to grab it, we should throw a new error
  if (!process.env.REACTAPPMYACCOUNTID || !process.env.REACTAPPMYPRIVATEKEY) {
    throw new Error("Environment variables REACTAPPMYACCOUNTID and
    REACTAPPMYPRIVATEKEY must be present");
  }

  // create your client
  const myAccountId = AccountId.fromString(process.env.REACTAPPMYACCOUNTID);
  const myPrivateKey = PrivateKey.fromString(process.env.REACTAPPMYPRIVATEKEY);

  const client = Client.forTestnet();
  client.setOperator(myAccountId, myPrivateKey);

  return (
    <Stack
      spacing={4}
      sx={{alignItems: 'center'}}
    >
      <Typography
        variant="h4"
        color="white"
      >
        Let's build a dApp on Hedera
      </Typography>
    </Stack>
  )
}
```

Build Your TransferTransaction

In our services folder, add a new file and name it hederaService.ts. This file will hold the code to send HBAR to our MetaMask wallet. Inside the file, let's create a function and name it sendHbar.

```
typescript
import { AccountId, Client, PrivateKey, TransactionReceiptQuery,
TransferTransaction } from "@hashgraph/sdk"

export const sendHbar = async (client:Client, fromAddress: AccountId | string,
toAddress: AccountId | string, amount: number, operatorPrivateKey: PrivateKey)
=> {
  const transferHbarTransaction = new TransferTransaction()
    .addHbarTransfer(fromAddress, -amount)
    .addHbarTransfer(toAddress, amount)
    .freezeWith(client);

  const transferHbarTransactionSigned = await
```

```

transferHbarTransaction.sign(operatorPrivateKey);
const transferHbarTransactionResponse = await
transferHbarTransactionSigned.execute(client);

// Get the child receipt or child record to return the Hedera Account ID for
the new account that was created
const transactionReceipt = await new TransactionReceiptQuery()
    .setTransactionId(transferHbarTransactionResponse.transactionId)
    .setIncludeChildren(true)
    .execute(client);

const childReceipt = transactionReceipt.children[0];

if(!childReceipt || childReceipt.accountId === null) {
    console.warn('No account id was found in child receipt. Child Receipt: $
{JSON.stringify(childReceipt, null, 4)}');
    return;
}

const newAccountId = childReceipt.accountId.toString();
console.log('Account ID of the newly created account: ${newAccountId}');
}

```

This function builds a TransferTransaction that will send a specified amount of HBAR from one account to another. In our case, our from account is our developer portal account, and our to account is our MetaMask wallet address.

Once the transaction is built, we approve it by signing it with our private key by calling `.sign()`. We call `.execute()` on our signed transaction using the client to pay for the transaction fee.

When a transaction is executed, the result is of type `TransactionResponse`. Use the transaction id from the `TransactionResponse` in a `TransactionReceiptQuery` to get a `TransactionReceipt`, including children transactions.

You can learn and read more about parent and child transactions on our [documentation site](#).

Note: For security purposes, the account sending the tokens should be on a backend server, but for simplicity, it will be on the frontend. This is a reminder that private keys should never be exposed on the frontend, as that is the easiest way to lose control of your account.

Send HBAR to MetaMask Wallet

We've written the code necessary to connect our application to MetaMask and installed the Hedera JavaScript SDK. Now it's time to focus on connecting all the parts and sending HBAR to our MetaMask wallet.

Configure State Management

React has a feature called Context that allows you to easily pass data between components without prop drilling. We will be leveraging this feature to save the MetaMask wallet address and enable us to access it from various components in our react-app.

Let's edit our `src/contexts/GlobalAppContext.tsx` file, which came with our template, to look like this:

```

typescript
import { createContext, ReactNode, useState } from "react";

```

```

const defaultValue = {
  metamaskAccountAddress: '',
  setMetamaskAccountAddress: (newValue: string) => { },
}

export const GlobalAppContext = createContext(defaultValue)

export const GlobalAppContextProvider = (props: { children: ReactNode |
undefined }) => {
  const [metamaskAccountAddress, setMetamaskAccountAddress] = useState('')

  return (
    <GlobalAppContext.Provider
      value={{
        metamaskAccountAddress,
        setMetamaskAccountAddress
      }}
    >
      {props.children}
    </GlobalAppContext.Provider>
  )
}

```

Add Functionality to the NavBar Button

Once we've set up our context, we use it to share the application state throughout the app. We use the context by calling `useContext()` and passing in `GlobalAppContext` as an argument. This allows us to get and set the wallet address from anywhere in the app.

Add the following code to the top of the file `src/components/Navbar.tsx`:

```

typescript
export default function NavBar() {
  // use the GlobalAppContext to keep track of the metamask account connection
  const { metamaskAccountAddress, setMetamaskAccountAddress } =
    useContext(GlobalAppContext);

```

Next, we will create a new function to connect to MetaMask and store the wallet address in the `GlobalAppContext`.

```

typescript
const retrieveWalletAddress = async () => {
  const addresses = await connectToMetamask();
  if (addresses) {
    // grab the first wallet address
    setMetamaskAccountAddress(addresses[0]);
    console.log(addresses[0]);
  }
}

```

Now we can add an `onClick` to our button and change the text to say 'Connect to MetaMask'. `onClick` will call `retrieveWalletAddress`.

Your completed button code will look like this:

```

typescript
<Button
  variant='contained'
  color='secondary'

```

```

    sx={{
      ml: 'auto'
    }}
    onClick={retrieveWalletAddress}
  >
    {metamaskAccountAddress === "" ?
      "Connect to MetaMask" :
      Connected to: ${metamaskAccountAddress.substring(0, 8)}...}
</Button>

```

Add Send HBAR Button

In the Home.tsx file, we will call our global context to gain access to our metamaskAccountAddress state variable.

```

typescript
export default function Home() {
  const { metamaskAccountAddress } = useContext(GlobalAppContext);

```

Next, add a new button to send HBAR to our MetaMask wallet. After the closing tag `</Typography>`, create a material UI button and add `onClick`. The `onClick` will call `sendHbar()`, which is inside `src/services/hederaService.ts`.

```

typescript
onClick={() => {
  sendHbar(client, myAccountId, AccountId.fromEvmAddress(0, 0,
    metamaskAccountAddress), 7, myPrivateKey)
}}

```

The to address will be the `metamaskAccountAddress` that we pulled out from our context using `useContext(GlobalAppContext)`.

Your completed button code will look like this:

```

typescript
<Stack
  spacing={4}
  sx={{alignItems: 'center'}}
>
  <Typography
    variant="h4"
    color="white"
  >
    Let's build a dApp on Hedera
  </Typography>
  <Button
    variant="contained"
    color="secondary"
    onClick={() => {
      sendHbar(client, myAccountId, AccountId.fromEvmAddress(0, 0,
        metamaskAccountAddress), 7, myPrivateKey)
    }}
  >
    Transfer HBAR to MetaMask Account
  </Button>
</Stack>

```

We're ready to run our application! Open a terminal In the root directory of the project and run:

```
bash
npm run start
```

Once up and running, click on the 'Connect to MetaMask' button in the upper right-hand corner.

```
<figure><figcaption></figcaption></figure>
```

After you click on it, the MetaMask pop-up window will open, and you will be asked to switch from your previously connected network to the Hedera Testnet.

```
<figure><figcaption></figcaption></figure>
```

Choose the account that you would like to connect to this application.

Debugging tip: If you have previously connected your account to this dApp and clicking connect is not opening the wallet, disconnect all the connected accounts and then try again\

```
<figure><figcaption></figcaption></figure>
```

Once connected, send HBAR by clicking on the 'SEND HBAR TO METAMASK' button. You can open your console and see that the transaction response is printed to the console.

```
<figure><figcaption></figcaption></figure>
```

Summary

You learned how to build a transfer transaction that sends an amount of HBAR through the Hedera Testnet to a MetaMask account. This can also be applied to other applications, and I encourage all to keep building.

Congratulations! 🎉 You successfully followed the tutorial to create an HBAR faucet for MetaMask and a Hedera React application that integrates with MetaMask! Feel free to reach out in Discord if you have any questions!

Additional Resources

- ☞ TypeScript Project Repository
- ☞ JavaScript Project Repository
- ☞ Download MetaMask Wallet
- ☞ Have a question? Ask on StackOverflow

```
<table data-card-size="large" data-view="cards"><thead><tr><th>
```

align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Abi, DevRel Engineer</p><p>GitHub LinkedIn</p></td><td align="center"><p>Editor: Krystal, Technical Writer</p><p>GitHub Twitter</p></td></tr></tbody></table>
--

deploy-a-contract-using-the-hedera-token-service.md:

Deploy a Contract Using the Hedera Token Service

Summary

In this example, you will learn how to create a Solidity contract that interacts with the Hedera Token Service (HTS). The initial release of this feature supports token mint, burn, associate, dissociate, and transfer transactions.

The example does not cover the environment setup or creating certain variables that may be seen in the code blocks. The full coding example can be found at the end of the page.

```
{% hint style="warning" %}
Smart contract entity auto renewal and expiry will be enabled in a future
release. Please check out HIP-16 for more information.
{% endhint %}
```

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

1. Create Your "HTS" Smart Contract

In this example, you will associate a token to an account and transfer tokens to the associated account by interacting with the HTS contract deployed to Hedera. The HTS contract has three functions that allow you to associate, transfer, and dissociate tokens from a Hedera account.

```
<mark style="color:blue;">tokenAssociate</mark>
<mark style="color:blue;">tokenTransfer</mark>
<mark style="color:blue;">tokenDissociate</mark>
```

The HTS.sol will serve as a reference to the contract that was compiled. The HTS.json file contains the `<mark style="color:blue;">data.bytecode.object</mark>` field that will be used to store the contract bytecode in a file on the Hedera network.

To write a contract using HTS, you will need to add the HTS Solidity support libraries to your project and import them into your contract. Please see the HTS.sol example for reference. The IHederaTokenService.sol will need to be in

the same directory as the other two files. An explanation of the functions can be found [here](#).

```
HederaTokenService.sol
HederaResponseCodes.sol
IHederaTokenService.sol
```

```
{% tabs %}
{% tab title="HTS.sol" %}
solidity
// SPDX-License-Identifier: Apache-2.0
pragma solidity ^0.6.12;
```

```
import "./HederaTokenService.sol";
import "./HederaResponseCodes.sol";
```

```
contract HTS is HederaTokenService {
```

```
    function tokenAssociate(address sender, address tokenAddress) external {
        int response = HederaTokenService.associateToken(sender, tokenAddress);

        if (response != HederaResponseCodes.SUCCESS) {
            revert ("Associate Failed");
        }
    }
```

```
    function tokenTransfer(address tokenId, address fromAccountId , address
toAccountId , int64 tokenAmount) external {
        int response = HederaTokenService.transferToken(tokenId, fromAccountId,
toAccountId, tokenAmount);

        if (response != HederaResponseCodes.SUCCESS) {
            revert ("Transfer Failed");
        }
    }
```

```
    function tokenDissociate(address sender, address tokenAddress) external {
        int response = HederaTokenService.dissociateToken(sender, tokenAddress);

        if (response != HederaResponseCodes.SUCCESS) {
            revert ("Dissociate Failed");
        }
    }
}
```

```
{% endtab %}
```

```
{% tab title="HTS.json" %}
json
{
  "deploy": {
    "VM:-": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "main:1": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "ropsten:3": {
      "linkReferences": {},
      "autoDeployLib": true
    },
  },
}
```

[illegible]

[illegible]

0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH4 0xECA36917 PUSH1 0xE0 SHL
DUP9 DUP9 DUP9 DUP9 PUSH1 0x40 MLOAD PUSH1 0x24 ADD DUP1 DUP6 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD DUP5
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1
0x20 ADD DUP3 PUSH1 0x7 SIGNEXTEND DUP2 MSTORE PUSH1 0x20 ADD SWAP5 POP POP
POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2 MSTORE SWAP1 PUSH1
0x40 MSTORE SWAP1 PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x46B JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x448 JUMP JUMPDEST PUSH1 0x1 DUP4 PUSH1
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1
DUP3 OR DUP6 MSTORE POP POP POP POP POP POP SWAP1 POP ADD SWAP2 POP POP PUSH1
0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0 DUP7 GAS CALL SWAP2 POP POP
RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x4CD JUMPI PUSH1 0x40 MLOAD SWAP2
POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE
RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD
RETURNDATACOPY PUSH2 0x4D2 JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP SWAP2
POP SWAP2 POP DUP2 PUSH2 0x4E3 JUMPI PUSH1 0x15 PUSH2 0x50A JUMP JUMPDEST DUP1
DUP1 PUSH1 0x20 ADD SWAP1 MLOAD PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x4F8 JUMPI
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20
ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP JUMPDEST PUSH1 0x3 SIGNEXTEND SWAP3 POP
POP POP SWAP5 SWAP4 POP POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x60
PUSH2 0x167 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH4
0x99794E8 PUSH1 0xE0 SHL DUP7 DUP7 PUSH1 0x40 MLOAD PUSH1 0x24 ADD DUP1 DUP4
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
DUP3 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1
0x20 ADD SWAP3 POP POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2
MSTORE SWAP1 PUSH1 0x40 MSTORE SWAP1 PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x614 JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x5F1 JUMP JUMPDEST PUSH1 0x1 DUP4 PUSH1
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1
DUP3 OR DUP6 MSTORE POP POP POP POP POP POP SWAP1 POP ADD SWAP2 POP POP PUSH1
0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0 DUP7 GAS CALL SWAP2 POP POP
RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x676 JUMPI PUSH1 0x40 MLOAD SWAP2
POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE
RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD
RETURNDATACOPY PUSH2 0x67B JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP SWAP2
POP SWAP2 POP DUP2 PUSH2 0x68C JUMPI PUSH1 0x15 PUSH2 0x6B3 JUMP JUMPDEST DUP1
DUP1 PUSH1 0x20 ADD SWAP1 MLOAD PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x6A1 JUMPI
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20
ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP JUMPDEST PUSH1 0x3 SIGNEXTEND SWAP3 POP
POP POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x60 PUSH2 0x167
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH4 0x49146BDE PUSH1
0xE0 SHL DUP7 DUP7 PUSH1 0x40 MLOAD PUSH1 0x24 ADD DUP1 DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD DUP3
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
SWAP3 POP POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2 MSTORE
SWAP1 PUSH1 0x40 MSTORE SWAP1 PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x7BB JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP

```
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x798 JUMP JUMPDEST PUSH1 0x1DUP4 PUSH1  
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1  
DUP3 OR DUP6 MSTORE POP POP POP POP POP POP SWAP1 POP ADD SWAP2 POP POP PUSH1  
0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0 DUP7 GAS CALL SWAP2 POP POP  
RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x81D JUMPI PUSH1 0x40 MLOAD SWAP2  
POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE  
RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD  
RETURNDATACOPY PUSH2 0x822 JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP SWAP2  
POP SWAP2 POP DUP2 PUSH2 0x833 JUMPI PUSH1 0x15 PUSH2 0x85A JUMP JUMPDEST DUP1  
DUP1 PUSH1 0x20 ADD SWAP1 MLOAD PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x848 JUMPI  
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20  
ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP POP JUMPDEST PUSH1 0x3 SIGNEXTEND SWAP3 POP  
POP POP SWAP3 SWAP2 POP POP JUMP INVALID LOG2 PUSH5 0x6970667358 0x22 SLT  
KECCAK256 PUSH9 0xBA1095E27DFAF338E8 0xEE SWAP14 MULMOD EQ RETURN 0x28 INVALID  
PUSH11 0x23627CE5B8245B5FD09275 0xBA PUSH23  
0xD964736F6C634300060C003300000000000000000000 " ,  
    "sourceMap": "138:923:0:-:0;;;;;;;;;;;;;;",  
},  
"deployedBytecode": {  
    "immutableReferences": {},  
    "linkReferences": {},  
    "object":  
"608060405234801561001057600080fd5b50600436106100415760003560e01c80633a04033c146  
100465780634753b51b146100d75780637f6314d01461013b575b600080fd5b6100d560048036036  
08081101561005c57600080fd5b81019080803573fffffffffffffffffffffffffffffffffffff  
f169060200190929190803573fffffffffffffffffffffffffffffffffffffffffffff169060200190929  
190803573fffffffffffffffffffffffffffffffffffffffffffff169060200190929190803560070b906  
020019092919050505061019f565b005b610139600480360360408110156100ed57600080fd5b810  
19080803573fffffffffffffffffffffffffffffffffffff169060200190929190803573ffffff  
fffffffffffffffffffffffffffffffffffff16906020019092919050505061022f565b005b61019d6  
004803603604081101561015157600080fd5b81019080803573ffffffffffffffffffffffffffff  
fffffffffffff169060200190929190803573fffffffffffffffffffffffffffffffffffff16906  
02001909291905050506102bb565b005b60006101ad85858585610347565b9050601660030b81146  
10228576040517f08c379a0000000000000000000000000000000000000000000000000000000  
15260040180806020018281038252600f8152602001807f5472616e73666572204661696c6564000  
000000000000000000000000000000081525060200191505060405180910390fd5b5050505050565  
b600061023b8383610519565b9050601660030b81146102b6576040517f08c379a0000000000000  
000000000000000000000000000000000000000000000008152600401808060200182810382526011815  
2602001807f446973736f6369617465204661696c656400000000000000000000000000000000081525  
060200191505060405180910390fd5b505050565b60006102c783836106c0565b9050601660030b8  
114610342576040517f08c379a000000000000000000000000000000000000000000000000000  
00081526004018080602001828103825260108152602001807f4173736f6369617465204661696c6  
56400000000000000000000000000000000000000000000081525060200191505060405180910390fd5b505050565  
b600080606061016773fffffffffffffffffffffffffffffffffffff1663eca3691760e01b888  
88888604051602401808573fffffffffffffffffffffffffffff1681526020018473f  
fffffffffffffffffffffffffffffffffffff1681526020018373fffffffffffffffffffff  
fffffffffffff1681526020018260070b815260200194505050505060405160208183030381529  
0604052907bfffffffffffffffffffffffffffffffffffff1916602082018  
0517bfffffffffffffffffffffffffffffffffffff8381831617835250505  
0506040518082805190602001908083835b6020831061046b5780518252602082019150602081019  
050602083039250610448565b6001836020036101000a03801982511681845116808217855250505  
05050509050019150506000604051808303816000865af19150503d80600081146104cd576040519  
150601f19603f3d011682016040523d82523d6000602084013e6104d2565b606091505b509150915  
0816104e357601561050a565b8080602001905160208110156104f857600080fd5b8101908080519  
0602001909291905050505b60030b92505050949350505050565b600080606061016773ffffff  
fffffffffffff1663099794e860e01b8686604051602401808373ffffff  
fffffffffffff1681526020018273ffffff  
fffffffffffff16815260200192505050604051602081830303815290604052907bffffff  
fffffffffffff19166020820180517bffffff  
fffffffffffff83818316178352505050604051808280519060200190808  
3835b6020831061061457805182526020820191506020810190506020830392506105f1565b60018  
36020036101000a038019825116818451168082178552505050505090500191505060006040518  
08303816000865af19150503d8060008114610676576040519150601f19603f3d011682016040523  
d82523d6000602084013e61067b565b606091505b50915091508161068c5760156106b3565b80806
```

```
02001905160208110156106a157600080fd5b81019080805190602001909291905050505b60030b9
250505092915050565b600080606061016773ffffffffffffffffffffffffffffffffffffffff166
349146bde60e01b8686604051602401808373ffffffffffffffffffffffffffffffffffffffff168
1526020018273ffffffffffffffffffffffffffffffffffffffff168152602001925050506040516
02081830303815290604052907bffffffffffffffffffffffffffffffffffffffffffffffffffffff
fff19166020820180517bfffffffffffffffffffffffffffffffffffffffffffffffffffffffffff838
18316178352505050506040518082805190602001908083835b602083106107bb578051825260208
2019150602081019050602083039250610798565b6001836020036101000a0380198251168184511
680821785525050505050509050019150506000604051808303816000865af19150503d806000811
461081d576040519150601f19603f3d011682016040523d82523d6000602084013e610825265b606
091505b509150915081601803357601561085a565b80806020019051602081101561848257600080f
d5b81019080805190602001909291905050505b60030b925050509291505056fea26469706673582
2122068ba1095e27dfaf338e8ee9d0914f328fe6a23627ce5b8245b5fd09275ba76d964736f6c634
300060c0033",
```

0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD DUP5
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1
0x20 ADD DUP3 PUSH1 0x7 SIGNEXTEND DUP2 MSTORE PUSH1 0x20 ADD SWAP5 POP POP POP
POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2 MSTORE SWAP1 PUSH1
0x40 MSTORE SWAP1 PUSH28
0xFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x46B JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x448 JUMP JUMPDEST PUSH1 0x1 DUP4 PUSH1
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1
DUP3 OR DUP6 MSTORE POP POP POP POP POP POP SWAP1 POP ADD SWAP2 POP POP PUSH1
0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0 DUP7 GAS CALL SWAP2 POP POP
RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x4CD JUMPI PUSH1 0x40 MLOAD SWAP2
POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE
RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD
RETURNDATACOPY PUSH2 0x4D2 JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP SWAP2
POP SWAP2 POP DUP2 PUSH2 0x4E3 JUMPI PUSH1 0x15 PUSH2 0x50A JUMP JUMPDEST DUP1
DUP1 PUSH1 0x20 ADD SWAP1 MLOAD PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x4F8 JUMPI
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20
ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP JUMPDEST PUSH1 0x3 SIGNEXTEND SWAP3 POP
POP POP SWAP5 SWAP4 POP POP POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x60
PUSH2 0x167 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH4
0x99794E8 PUSH1 0xE0 SHL DUP7 DUP7 PUSH1 0x40 MLOAD PUSH1 0x24 ADD DUP1 DUP4
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
DUP3 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1
0x20 ADD SWAP3 POP POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2
MSTORE SWAP1 PUSH1 0x40 MSTORE SWAP1 PUSH28
0xFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x614 JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x5F1 JUMP JUMPDEST PUSH1 0x1 DUP4 PUSH1
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1
DUP3 OR DUP6 MSTORE POP POP POP POP POP POP SWAP1 POP ADD SWAP2 POP POP PUSH1
0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 PUSH1 0x0 DUP7 GAS CALL SWAP2 POP POP
RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x676 JUMPI PUSH1 0x40 MLOAD SWAP2
POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE
RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD
RETURNDATACOPY PUSH2 0x67B JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP SWAP2
POP SWAP2 POP DUP2 PUSH2 0x68C JUMPI PUSH1 0x15 PUSH2 0x6B3 JUMP JUMPDEST DUP1
DUP1 PUSH1 0x20 ADD SWAP1 MLOAD PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x6A1 JUMPI
PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20
ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP JUMPDEST PUSH1 0x3 SIGNEXTEND SWAP3 POP
POP POP SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 PUSH1 0x60 PUSH2 0x167
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH4 0x49146BDE PUSH1
0xE0 SHL DUP7 DUP7 PUSH1 0x40 MLOAD PUSH1 0x24 ADD DUP1 DUP4 PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD DUP3
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD
SWAP3 POP POP POP PUSH1 0x40 MLOAD PUSH1 0x20 DUP2 DUP4 SUB SUB DUP2 MSTORE
SWAP1 PUSH1 0x40 MSTORE SWAP1 PUSH28
0xFF NOT AND PUSH1 0x20
DUP3 ADD DUP1 MLOAD PUSH28
0xFF DUP4 DUP2 DUP4 AND OR
DUP4 MSTORE POP POP POP POP PUSH1 0x40 MLOAD DUP1 DUP3 DUP1 MLOAD SWAP1 PUSH1
0x20 ADD SWAP1 DUP1 DUP4 DUP4 JUMPDEST PUSH1 0x20 DUP4 LT PUSH2 0x7BB JUMPI DUP1
MLOAD DUP3 MSTORE PUSH1 0x20 DUP3 ADD SWAP2 POP PUSH1 0x20 DUP2 ADD SWAP1 POP
PUSH1 0x20 DUP4 SUB SWAP3 POP PUSH2 0x798 JUMP JUMPDEST PUSH1 0x1 DUP4 PUSH1
0x20 SUB PUSH2 0x100 EXP SUB DUP1 NOT DUP3 MLOAD AND DUP2 DUP5 MLOAD AND DUP1

[illegible]

```

        "internalType": "address",
        "name": "sender",
        "type": "address"
    },
    {
        "internalType": "address",
        "name": "tokenAddress",
        "type": "address"
    }
],
"name": "tokenAssociate",
"outputs": [],
"stateMutability": "nonpayable",
"type": "function"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "sender",
            "type": "address"
        },
        {
            "internalType": "address",
            "name": "tokenAddress",
            "type": "address"
        }
    ],
    "name": "tokenDissociate",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "tokenId",
            "type": "address"
        },
        {
            "internalType": "address",
            "name": "fromAccountId",
            "type": "address"
        },
        {
            "internalType": "address",
            "name": "toAccountId",
            "type": "address"
        },
        {
            "internalType": "int64",
            "name": "tokenAmount",
            "type": "int64"
        }
    ],
    "name": "tokenTransfer",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
}
]
}

```

```
{% endtab %}  
{% endtabs %}
```

2. Store the Smart Contract Bytecode on Hedera

Create a file using the `FileCreateTransaction()` API to store the hex-encoded byte code of the "HTS" contract. Once the file is created, you can obtain the file ID from the receipt of the transaction.

```
{% hint style="warning" %}
```

Note: The bytecode is required to be hex-encoded. It should not be the actual data the hex represents.

```
{% endhint %}
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Import the HTS.json file from the resources folder
```

```
ClassLoader cl = HTS.class.getClassLoader();
```

```
Gson gson = new Gson();
```

```
JsonObject jsonObject;
```

```
//Get the json file
```

```
InputStream jsonStream = cl.getResourceAsStream("HTS.json");
```

```
jsonObject = gson.fromJson(new InputStreamReader(jsonStream,  
StandardCharsets.UTF8), JsonObject.class);
```

```
//Store the "object" field from the HTS.json file as hex-encoded bytecode
```

```
String object =
```

```
jsonObject.getAsJsonObject("data").getAsJsonObject("bytecode").get("object").get  
AsString();
```

```
byte[] bytecode = object.getBytes(StandardCharsets.UTF8);
```

```
//Create a file on Hedera and store the hex-encoded bytecode
```

```
FileCreateTransaction fileCreateTx = new FileCreateTransaction()  
    .setKeys(privateKeyTest)  
    .setContents(bytecode);
```

```
//Submit the file to the Hedera test network
```

```
TransactionResponse submitTx = fileCreateTx.execute(client);
```

```
//Get the receipt of the file create transaction
```

```
TransactionReceipt fileReceipt = submitTx.getReceipt(client);
```

```
//Get the file ID
```

```
FileId newFileId = fileReceipt.fileId;
```

```
//Log the file ID
```

```
System.out.println("The smart contract byte code file ID is " + newFileId);
```

```
//v2.6.0 Hedera Java SDK
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Get the contract bytecode
```

```
const bytecode = htsContract.data.bytecode.object;
```

```
//Create a file on Hedera and store the hex-encoded bytecode
```



```

const fileCreateTx = new FileCreateTransaction()
    .setContents(bytecode);

//Submit the file to the Hedera test network signing with the transaction fee
payer key specified with the client
const submitTx = await fileCreateTx.execute(client);

//Get the receipt of the file create transaction
const fileReceipt = await submitTx.getReceipt(client);

//Get the file ID from the receipt
const bytecodeFileId = fileReceipt.fileId;

//Log the file ID
console.log("The smart contract bytecode file ID is " +bytecodeFileId)

{% endtab %}

{% tab title="Go" %}
go
//Get the HTS contract bytecode
rawSmartContract, err := ioutil.ReadFile("./hts.json")
if err != nil {
    println(err.Error(), ": error reading hts.json")
    return
}

var contract contract = contract{}

err = json.Unmarshal([]byte(rawSmartContract), &contract)
if err != nil {
    println(err.Error(), ": error unmarshaling the json file")
    return
}

smartContractByteCode := []byte(contract.Object)

// Upload a file containing the byte code
fileCreateTx, err := hedera.NewFileCreateTransaction().
    SetContents([]byte(smartContractByteCode)).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating file")
    return
}

//Get the receipt of the transaction
fileTxReceipt, err := fileCreateTx.GetReceipt(client)
if err != nil {
    println(err.Error(), ": error getting file create transaction receipt")
    return
}

//Get the bytecode file ID
byteCodeFileID := fileTxReceipt.FileID

fmt.Printf("The contract bytecode file ID: %v\n", byteCodeFileID)

{% endtab %}
{% endtabs %}

```

3. Deploy a Hedera Smart Contract

Create the contract and set the file ID to the file that contains the hex-encoded bytecode from the previous step. You will need to set the gas high enough to deploy the contract. The gas should be estimated to be within 25% of the actual gas cost to avoid paying extra gas. You can read more about gas and fees [here](#).

```
{% hint style="warning" %}
```

Note: You will need to set the gas value high enough to deploy the contract. If you don't have enough gas, you will receive an `<mark style="color:blue;">INSUFFICIENTGAS</mark>` response. If you set the value too high you will be refunded a maximum of 20% of the amount that was set for the transaction.

```
{% endhint %}
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//Deploy the contract
```

```
ContractCreateTransaction contractTx = new ContractCreateTransaction()
```

```
    //The contract bytecode file
```

```
    .setBytecodeFileId(newFileId)
```

```
    //The max gas to reserve for this transaction
```

```
    .setGas(20000000);
```

```
//Submit the transaction to the Hedera test network
```

```
TransactionResponse contractResponse = contractTx.execute(client);
```

```
//Get the receipt of the file create transaction
```

```
TransactionReceipt contractReceipt = contractResponse.getReceipt(client);
```

```
//Get the smart contract ID
```

```
ContractId newContractId = contractReceipt.contractId;
```

```
//Log the smart contract ID
```

```
System.out.println("The smart contract ID is " + newContractId);
```

```
//v2.6.0 Hedera Java SDK
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//Deploy the contract instance
```

```
const contractTx = await new ContractCreateTransaction()
```

```
    //The bytecode file ID
```

```
    .setBytecodeFileId(bytecodeFileId)
```

```
    //The max gas to reserve
```

```
    .setGas(20000000);
```

```
//Submit the transaction to the Hedera test network
```

```
const contractResponse = await contractTx.execute(client);
```

```
//Get the receipt of the file create transaction
```

```
const contractReceipt = await contractResponse.getReceipt(client);
```

```
//Get the smart contract ID
```

```
const newContractId = contractReceipt.contractId;
```

```
//Log the smart contract ID
```

```
console.log("The smart contract ID is " + newContractId);
```

```
{% endtab %}
```

```

{% tab title="Go" %}
go
// Deploy the contract instance
contractTransactionID, err := hedera.NewContractCreateTransaction().
    //The max gas for the transaction
    SetGas(20000000).
    //The contract bytecode file ID
    SetBytecodeFileID(byteCodeFileID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating contract")
    return
}

//Get the contract receipt
contractReceipt, err := contractTransactionID.GetReceipt(client)

//Get the contract contract ID
contractId := contractReceipt.ContractID

//Log the contract ID
fmt.Printf("The contract ID %v\n", contractId)

{% endtab %}
{% endtabs %}

```

4. Call the `tokenAssociate` Contract Function

The `tokenAssociate` function in the contract was previously used to associate tokens created with the Hedera Token Service (HTS). However, due to a change in the security model, it is no longer possible to associate HTS tokens using this function. Instead, you should use the Hedera SDK to perform token associations. You will pass the token ID and account ID to the function. The parameters must be provided in the order expected by the function to execute successfully.

```

{% tabs %}
{% tab title="Java" %}
java
//Associate the token to an account using the HTS contract
TokenAssociateTransaction transaction = new TokenAssociateTransaction()
    .setAccountId(accountIdTest)
    .setTokenId(Collections.singletonList(tokenId))
    .freezeWith(client);

//Sign with the account key to associate and submit to the Hedera network
TransactionResponse associateTokenResponse =
transaction.sign(privateKeyTest).execute(client);

System.out.println("The transaction status: "
+associateTokenResponse.getReceipt(client).status);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Associate the token to an account using the SDK
const transaction = new TokenAssociateTransaction()
    .setAccountId(accountIdTest)
    .setTokenIds([tokenId])

```

```

    .freezeWith(client);

//Sign the transaction with the client
const signTx = await transaction.sign(accountKeyTest);

//Submit the transaction
const submitAssociateTx = await signTx.execute(client);

//Get the receipt
const txReceipt = await submitAssociateTx.getReceipt(client);

//Get transaction status
const txStatus = txReceipt.status;

console.log("The associate transaction was " + txStatus.toString());

{% endtab %}

{% tab title="Go" %}
go
//Associate an account with a token
associateTx, err := hedera.NewTokenAssociateTransaction().
    SetAccountID(accountIdTest).
    SetTokenIDs(tokenId).
    FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign with the private key of the account that is being associated to a token,
submit the transaction to a Hedera network
associateTxResponse, err := associateTx.Sign(privateKeyTest).Execute(client)

if err != nil {
    panic(err)
}

//Get the receipt
associateTxReceipt, err := associateTxResponse.GetReceipt(client)

if err != nil {
    panic(err)
}

//Get transaction status
txStatus := associateTxReceipt.Status

fmt.Printf("The associate transaction status %v\n", txStatus)

{% endtab %}
{% endtabs %}

```

5. Call the `approveTokenAllowance` Function

Using the `approveTokenAllowance` function is a crucial step before initiating a transfer with a smart contract on Hedera. This function grants the necessary permissions from the token owner to authorize the transfer. It serves as a stringent access control measure, ensuring that only approved contracts or accounts can spend the designated tokens. You will pass the owner which is the account that owns the fungible tokens and grants the allowance to the spender, the spender who is the account authorized by the owner to spend fungible tokens

from the owner's account. The spender covers the transaction fees for token transfers. And the amount which is the number of tokens the spender is authorized to spend from the owner's account.

```
{% tabs %}
{% tab title="Java" %}
java
// Convert the contract ID to an account ID
AccountId contractIdAsAccountId =
AccountId.fromString(newContractId.toString());

//Approve the token allowance
AccountAllowanceApproveTransaction transaction = new
AccountAllowanceApproveTransaction()
    .approveHbarAllowance(treasuryAccountId, newContractId, Hbar.from(5));

//Sign the transaction with the owner account key and the transaction fee payer
key (client)
TransactionResponse txResponse =
transaction.freezeWith(client).sign(treasuryKey).execute(client);

//Request the receipt of the transaction
TransactionReceipt receipt = txResponse.getReceipt(client);

//Get the transaction consensus status
Status transactionStatus = receipt.status;

System.out.println("The transaction consensus status for the allowance function
is " +transactionStatus);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Approve the token allowance
const transactionAllowance = new AccountAllowanceApproveTransaction()
    .approveTokenAllowance(tokenId, treasuryAccountId, newContractId, 5)
    .freezeWith(client);

//Sign the transaction with the owner account key
const signTxAllowance = await transactionAllowance.sign(treasuryKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponseAllowance = await signTxAllowance.execute(client);

//Request the receipt of the transaction
const receiptAllowance = await txResponseAllowance.getReceipt(client);

//Get the transaction consensus status
const transactionStatusAllowance = receiptAllowance.status;

console.log(
    "The transaction consensus status for the allowance function is " +
    transactionStatusAllowance.toString()
);

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction
transaction := hedera.NewAccountAllowanceApproveTransaction().
    ApproveHbarAllowance(ownerAccount, spenderAccountId, Hbar.fromTinybars(500)
```

```

        FreezeWith(client)

if err != nil {
    panic(err)
}

//Sign the transaction with the owner account private key
txResponse, err := transaction.Sign(ownerAccountKey).Execute(client)

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the transaction consensus status
transactionStatus := receipt.Status

println("The transaction consensus status is ", transactionStatus)

{% endtab %}
{% endtabs %}

```

6. Call the `tokenTransfer` Contract Function

Transfer 100 units of the token to the account that was associated with the token. You will use the `ContractExecuteTransaction()` API and set the contract function to `tokenTransfer`. The contract function parameters must be provided in the order of the function expects to receive them.

The transaction must be signed by the account that is sending the tokens. In this case, it is the treasury account.

You can verify the transfer was successful by checking the account token balance!

```

{% tabs %}
{% tab title="Java" %}
java
//Transfer the new token to the account
//Contract function params need to be in the order of the paramters provided in
the tokenTransfer contract function
ContractExecuteTransaction tokenTransfer = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(2000000)
    .setFunction("tokenTransfer", new ContractFunctionParameters()
        //The ID of the token
        .addAddress(tokenId.toSolidityAddress())
        //The account to transfer the tokens from
        .addAddress(treasuryAccountId.toSolidityAddress())
        //The account to transfer the tokens to
        .addAddress(accountIdTest.toSolidityAddress())
        //The number of tokens to transfer
        .addInt64(5));

//Sign the token transfer transaction with the treasury account to authorize the
transfer and submit
ContractExecuteTransaction signTokenTransfer =
tokenTransfer.freezeWith(client).sign(treasuryKey);

//Submit transfer transaction

```

```

TransactionResponse submitTransfer = signTokenTransfer.execute(client);

//Get transaction status
Status txStatus = submitTransfer.getReceipt(client).status;

//Verify your account received the 5 tokens
AccountBalance newAccountBalance = new AccountBalanceQuery()
    .setAccountId(accountIdTest)
    .execute(client);

System.out.println("My new account balance is " +newAccountBalance.tokens);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Transfer the new token to the account
//Contract function params need to be in the order of the paramters provided in
the tokenTransfer contract function
const tokenTransfer = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(20000000)
    .setFunction("tokenTransfer", new ContractFunctionParameters()
        //The ID of the token
        .addAddress(tokenId.toSolidityAddress())
        //The account to transfer the tokens from
        .addAddress(treasuryAccountId.toSolidityAddress())
        //The account to transfer the tokens to
        .addAddress(accountIdTest.toSolidityAddress())
        //The number of tokens to transfer
        .addInt64(5));

//Sign the token transfer transaction with the treasury account to authorize the
transfer and submit
const signTokenTransfer = await
tokenTransfer.freezeWith(client).sign(treasuryKey);

//Submit transfer transaction
const submitTransfer = await signTokenTransfer.execute(client);

//Get transaction status
const transferTxStatus = await (await submitTransfer.getReceipt(client)).status;

//Get the transaction status
console.log("The transfer transaction status " +transferTxStatus.toString());

//Verify the account received the 5 tokens
const newAccountBalance = new AccountBalanceQuery()
    .setAccountId(accountIdTest)
    .execute(client);

console.log("My new account balance is " +(await
newAccountBalance).tokens.toString());

{% endtab %}

{% tab title="Go" %}
go
//Transfer the token
transferTx := hedera.NewContractExecuteTransaction().
    //The contract ID
    SetContractID(contractId).
    //The max gas
    SetGas(20000000).

```

```

//The contract function to call and parameters
SetFunction("tokenTransfer", contractParamsAmount)

//Sign with treasury key to authorize the transfer from the treasury account
signTx, err := transferTx.Sign(treasuryKey).Execute(client)

if err != nil {
    println(err.Error(), ": error executing contract")
    return
}
//Get the receipt
transferTxReceipt, err := signTx.GetReceipt(client)

if err != nil {
    println(err.Error(), ": error getting receipt")
    return
}

//Get transaction status
transferTxStatus := transferTxReceipt.Status

fmt.Printf("The transfer transaction status %v\n", transferTxStatus)

//Verify the transfer by checking the balance
transferAccountBalance, err := hedera.NewAccountBalanceQuery().
    SetAccountID(accountIdTest).
    Execute(client)

if err != nil {
    println(err.Error(), ": error getting balance")
    return
}

//Log the account token balance
fmt.Printf("The account token balance %v\n", transferAccountBalance.Tokens)

{% endtab %}
{% endtabs %}

{% hint style="info" %}
Note: Check out our smart contract mirror node rest APIs that return information
about a contract like contract results and logs!
{% endhint %}

```

Congratulations :tada:! You have learned how to deploy a contract using the Hedera Token Service and completed the following:

- Associated an HTS token by using the SDK
- Approved the token allowance so that the contract can transfer tokens
- Transferred tokens using the deployed contract

Code Check 

<details>

<summary>Java</summary>

```

java
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.hedera.hashgraph.sdk.;

```



```

import io.github.cdimascio.dotenv.Dotenv;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeoutException;

public class HTS {

    public static void main(String[] args) throws TimeoutException,
        PrecheckStatusException, ReceiptStatusException, InterruptedException,
        IOException {

        AccountId accountIdTest =
        AccountId.fromString(Dotenv.load().get("MYACCOUNTID"));
        PrivateKey privateKeyTest =
        PrivateKey.fromString(Dotenv.load().get("MYPRIVATEKEY"));

        Client client = Client.forTestnet();
        client.setOperator(accountIdTest, privateKeyTest);

        //Import the HTS.json file from the resources folder
        ClassLoader cl = HTS.class.getClassLoader();

        Gson gson = new Gson();
        JsonObject jsonObject;

        //Get the json file
        InputStream jsonStream = cl.getResourceAsStream("HTS.json");
        jsonObject = gson.fromJson(new InputStreamReader(jsonStream,
        StandardCharsets.UTF8), JsonObject.class);

        //Store the "object" field from the HTS.json file as hex-encoded
        bytecode
        String object =
        jsonObject.getAsJsonObject("data").getAsJsonObject("bytecode").get("object").get
        AsString();
        byte[] bytecode = object.getBytes(StandardCharsets.UTF8);

        //Create a treasury Key
        PrivateKey treasuryKey = PrivateKey.generateED25519();

        //Create a treasury account
        AccountCreateTransaction treasuryAccount = new
        AccountCreateTransaction()
            .setKey(treasuryKey)
            .setInitialBalance(new Hbar(10))
            .setAccountMemo("treasury account");

        //Submit the account create transaction
        TransactionResponse submitAccountCreateTx =
        treasuryAccount.execute(client);

        //Get the receipt of the transaction
        TransactionReceipt newAccountReceipt =
        submitAccountCreateTx.getReceipt(client);

        //Get the treasury account ID
        AccountId treasuryAccountId = newAccountReceipt.accountId;
        System.out.println("The new account ID is " +treasuryAccountId);

        //Create a token to interact with

```

```

TokenCreateTransaction createToken = new TokenCreateTransaction()
    .setTokenName("HSCS demo")
    .setTokenSymbol("H")
    .setTokenType(TokenType.FUNGIBLECOMMON)
    .setTreasuryAccountId(treasuryAccountId)
    .setInitialSupply(500);

//Submit the token create transaction
TransactionResponse submitTokenTx =
createToken.freezeWith(client).sign(treasuryKey).execute(client);

//Get the token ID
TokenId tokenId = submitTokenTx.getReceipt(client).tokenId;
System.out.println("The new token ID is " +tokenId);

//Create a file on Hedera and store the hex-encoded bytecode
FileCreateTransaction fileCreateTx = new FileCreateTransaction()
    .setKeys(privateKeyTest)
    .setContents(bytecode);

//Submit the file to the Hedera test network
TransactionResponse submitTx = fileCreateTx.execute(client);

//Get the receipt of the file create transaction
TransactionReceipt fileReceipt = submitTx.getReceipt(client);

//Get the file ID
FileId newFileId = fileReceipt.fileId;

//Log the file ID
System.out.println("The smart contract byte code file ID is " +
newFileId);

//Deploy the contract
ContractCreateTransaction contractTx = new ContractCreateTransaction()
    //The contract bytecode file
    .setBytecodeFileId(newFileId)
    //The max gas to reserve for this transaction
    .setGas(20000000);

//Submit the transaction to the Hedera test network
TransactionResponse contractResponse = contractTx.execute(client);

//Get the receipt of the file create transaction
TransactionReceipt contractReceipt =
contractResponse.getReceipt(client);

//Get the smart contract ID
ContractId newContractId = contractReceipt.contractId;

//Log the smart contract ID
System.out.println("The smart contract ID is " + newContractId);

//Associate the token to an account using the SDK
TokenAssociateTransaction transaction = new TokenAssociateTransaction()
    .setAccountId(accountIdTest)
    .setTokenId(Collections.singletonList(tokenId))
    .freezeWith(client)

//Sign with the account key to associate and submit to the Hedera
network

```

```

        TransactionResponse associateTokenResponse =
transaction.sign(privateKeyTest).execute(client);

        System.out.println("The transaction status: "
+associateTokenResponse.getReceipt(client).status);

        // Convert the contract ID to an account ID
        AccountId contractIdAsAccountId =
AccountId.fromString(newContractId.toString());

        //Approve the token allowance so that the contract can transfer tokens
from the treasury account
        AccountAllowanceApproveTransaction transaction = new
AccountAllowanceApproveTransaction()
            .approveTokenAllowance(tokenId, treasuryAccountId,
contractIdAsAccountId, 10);

        //Sign the transaction with the owner account key and the transaction
fee payer key (client)
        TransactionResponse txResponse =
transaction.freezeWith(client).sign(treasuryKey).execute(client);

        //Request the receipt of the transaction
        TransactionReceipt receipt = txResponse.getReceipt(client);

        //Get the transaction consensus status
        Status transactionStatus = receipt.status;

        System.out.println("The transaction consensus status is "
+transactionStatus);

        //Transfer the new token to the account
        //Contract function params need to be in the order of the paramters
provided in the tokenTransfer contract function
        ContractExecuteTransaction tokenTransfer = new
ContractExecuteTransaction()
            .setContractId(newContractId)
            .setGas(20000000)
            .setFunction("tokenTransfer", new ContractFunctionParameters()
                //The ID of the token
                .addAddress(tokenId.toSolidityAddress())
                //The account to transfer the tokens from
                .addAddress(treasuryAccountId.toSolidityAddress())
                //The account to transfer the tokens to
                .addAddress(accountIdTest.toSolidityAddress())
                //The number of tokens to transfer
                .addInt64(5);

        //Sign the token transfer transaction with the treasury account to
authorize the transfer and submit
        ContractExecuteTransaction signTokenTransfer =
tokenTransfer.freezeWith(client).sign(treasuryKey);

        //Submit transfer transaction
        TransactionResponse submitTransfer = signTokenTransfer.execute(client);

        //Get transaction status
        Status txStatus = submitTransfer.getReceipt(client).status;

        //Get the transaction status
        System.out.println("The transfer transaction status " +txStatus);

        //Verify your account received the 5 tokens
        AccountBalance newAccountBalance = new AccountBalanceQuery()

```

```

        .setAccountId(accountIdTest)
        .execute(client);

        System.out.println("My new account balance is "
+newAccountBalance.tokens);
    }
}

</details>

<details>

<summary>JavaScript</summary>

javascript
require("dotenv").config();

const {
  Hbar,
  Client,
  AccountId,
  TokenType,
  PrivateKey,
  AccountBalanceQuery,
  FileCreateTransaction,
  TokenCreateTransaction,
  ContractCreateTransaction,
  ContractExecuteTransaction,
  ContractFunctionParameters,
  AccountCreateTransaction,
  AccountAllowanceApproveTransaction,
  TokenAssociateTransaction,
} = require("@hashgraph/sdk");

// Import the compiled contract
const htsContract = require("./HTS.json");

async function htsContractFunction() {
  //Grab your Hedera testnet account ID and private key from your .env file
  const accountIdTest = AccountId.fromString(process.env.MYACCOUNTID);
  const accountKeyTest = PrivateKey.fromStringED25519(
    process.env.MYPRIVATEKEY
  );

  // If we weren't able to grab it, we should throw a new error
  if (accountIdTest == null || accountKeyTest == null) {
    throw new Error(
      "Environment variables myAccountId and myPrivateKey must be present"
    );
  }

  const client = Client.forTestnet();
  client.setOperator(accountIdTest, accountKeyTest);

  //Get the contract bytecode
  const bytecode = htsContract.data.bytecode.object;

  //Treasury Key
  const treasuryKey = PrivateKey.generateED25519();

  //Create token treasury account
  const treasuryAccount = new AccountCreateTransaction()
    .setKey(treasuryKey)

```

```

        .setInitialBalance(new Hbar(5))
        .setAccountMemo("treasury account");

//Submit the transaction to a Hedera network
const submitAccountCreateTx = await treasuryAccount.execute(client);

//Get the receipt of the transaction
const newAccountReceipt = await submitAccountCreateTx.getReceipt(client);

//Get the account ID from the receipt
const treasuryAccountId = newAccountReceipt.accountId;

console.log("The new account ID is " + treasuryAccountId);

//Create a token to interact with
const createToken = new TokenCreateTransaction()
    .setTokenName("HTS demo")
    .setTokenSymbol("H")
    .setTokenType(Token.Type.FungibleCommon)
    .setTreasuryAccountId(treasuryAccountId)
    .setInitialSupply(500);

//Sign with the treasury key
const signTokenTx = await createToken.freezeWith(client).sign(treasuryKey);

//Submit the transaction to a Hedera network
const submitTokenTx = await signTokenTx.execute(client);

//Get the token ID from the receipt
const tokenId = await (await submitTokenTx.getReceipt(client)).tokenId;

//Log the token ID
console.log("The new token ID is " + tokenId);

//Create a file on Hedera and store the hex-encoded bytecode
const fileCreateTx = new FileCreateTransaction().setContents(bytecode);

//Submit the file to the Hedera test network signing with the transaction fee
payer key specified with the client
const submitTx = await fileCreateTx.execute(client);

//Get the receipt of the file create transaction
const fileReceipt = await submitTx.getReceipt(client);

//Get the file ID from the receipt
const bytecodeFileId = fileReceipt.fileId;

//Log the file ID
console.log("The smart contract byte code file ID is " + bytecodeFileId);

//Deploy the contract instance
const contractTx = await new ContractCreateTransaction()
    //The bytecode file ID
    .setBytecodeFileId(bytecodeFileId)
    //The max gas to reserve
    .setGas(20000000);

//Submit the transaction to the Hedera test network
const contractResponse = await contractTx.execute(client);

//Get the receipt of the file create transaction
const contractReceipt = await contractResponse.getReceipt(client);

//Get the smart contract ID

```

```

const newContractId = contractReceipt.contractId;

//Log the smart contract ID
console.log("The smart contract ID is " + newContractId);

//Associate the token to an account using the SDK
const transaction = new TokenAssociateTransaction()
    .setAccountId(accountIdTest)
    .setTokenIds([tokenId])
    .freezeWith(client);

//Sign the transaction with the client
const signTx = await transaction.sign(accountKeyTest);

//Submit the transaction
const submitAssociateTx = await signTx.execute(client);

//Get the receipt
const txReceipt = await submitAssociateTx.getReceipt(client);

//Get transaction status
const txStatus = txReceipt.status;

console.log("The associate transaction was " + txStatus.toString());

//Approve the token allowance
const transactionAllowance = new AccountAllowanceApproveTransaction()
    .approveTokenAllowance(tokenId, treasuryAccountId, newContractId, 5)
    .freezeWith(client);

//Sign the transaction with the owner account key
const signTxAllowance = await transactionAllowance.sign(treasuryKey);

//Sign the transaction with the client operator private key and submit to a
Hedera network
const txResponseAllowance = await signTxAllowance.execute(client);

//Request the receipt of the transaction
const receiptAllowance = await txResponseAllowance.getReceipt(client);

//Get the transaction consensus status
const transactionStatusAllowance = receiptAllowance.status;

console.log(
    "The transaction consensus status for the allowance function is " +
    transactionStatusAllowance.toString()
);

//Transfer the new token to the account
//Contract function params need to be in the order of the parameters provided
in the tokenTransfer contract function
const tokenTransfer = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(20000000)
    .setFunction(
        "tokenTransfer",
        new ContractFunctionParameters()
            //The ID of the token
            .addAddress(tokenId.toSolidityAddress())
            //The account to transfer the tokens from
            .addAddress(treasuryAccountId.toSolidityAddress())
            //The account to transfer the tokens to
            .addAddress(accountIdTest.toSolidityAddress())
            //The number of tokens to transfer

```

```

        .addInt64(5)
    );

    //Sign the token transfer transaction with the treasury account to authorize
the transfer and submit
    const signTokenTransfer = await tokenTransfer
        .freezeWith(client)
        .sign(treasuryKey);

    //Submit transfer transaction
    const submitTransfer = await signTokenTransfer.execute(client);

    //Get transaction status
    const transferTxStatus = await (
        await submitTransfer.getReceipt(client)
    ).status;

    //Get the transaction status
    console.log("The transfer transaction status " + transferTxStatus.toString());

    //Verify your account received the 10 tokens
    const newAccountBalance = new AccountBalanceQuery()
        .setAccountId(accountIdTest)
        .execute(client);

    console.log(
        "My new account balance is " + (await newAccountBalance).tokens.toString()
    );
}

void htsContractFunction();

```

</details>

<details>

<summary>Go</summary>

<pre class="language-go"><code class="lang-go">package main

```

import (
    "encoding/json"
    "fmt"
    "io/ioutil"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

```

```

type contract struct {
    // ignore the link references since it is empty
    Object      string json:"object"
    OpCodes     string json:"opcodes"
    SourceMap   string json:"sourceMap"
}

```

```

func main() {

```

```

    //Loads the .env file and throws an error if it cannot load the variables
from that file corectly
    err := godotenv.Load(".env")

```

```

    if err != nil {
        panic(fmt.Errorf("Unable to load enviroment variables from .env
file. Error:\n%\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    accountIdTest, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    privateKeyTest, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    //Create your testnet client
    client := hedera.ClientForTestnet()
    client.SetOperator(accountIdTest, privateKeyTest)

    //Treasury Key
    treasuryKey, err := hedera.PrivateKeyGenerateEd25519()

    //Create token treasury account
    treasuryAccount := hedera.NewAccountCreateTransaction().
        SetKey(treasuryKey).
        SetInitialBalance(hedera.NewHbar(5).
        SetAccountMemo("treasury account")

    //Submit the transaction to a Hedera network
    submitAccountCreateTx, err := treasuryAccount.Execute(client)

    //Get the receipt of the transaction
    newAccountReceipt, err := submitAccountCreateTx.GetReceipt(client)

    //Get the account ID from the receipt
    treasuryAccountId := newAccountReceipt.AccountID

    fmt.Printf("The treasury account ID: %v\n", treasuryAccountId)

    //Create a token to interact with
    createToken := hedera.NewTokenCreateTransaction().
        SetTokenName("HTS demo").
        SetTokenSymbol("H").
        SetTokenType(hedera.TokenTypeFungibleCommon).
        SetTreasuryAccountID(treasuryAccountId).
        SetInitialSupply(500)

    //Freeze the transaction for signing
    freezeTokenTx, err := createToken.FreezeWith(client)

    //Sign with the treasury key to authorize the transaction
    signTokenTx := freezeTokenTx.Sign(treasuryKey)

    //Submit the transaction
    submitTokenTx, err := signTokenTx.Execute(client)

    //Get the receipt of the transaction
    getTokenReceipt, err := submitTokenTx.GetReceipt(client)

    //Get the token ID
    tokenId := getTokenReceipt.TokenID

```



```

//Log the token ID
fmt.Printf("The token ID: %v\n", tokenId)

//Get the HTS contract bytecode
rawSmartContract, err := ioutil.ReadFile("./hts.json")
if err != nil {
    println(err.Error(), ": error reading hts.json")
    return
}

var contract contract = contract{}

err = json.Unmarshal([]byte(rawSmartContract), &contract)
if err != nil {
    println(err.Error(), ": error unmarshaling the json file")
    return
}

smartContractByteCode := []byte(contract.Object)

// Upload a file containing the byte code
fileCreateTx, err := hedera.NewFileCreateTransaction().
    SetContents([]byte(smartContractByteCode)).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating file")
    return
}

//Get the receipt of the transaction
fileTxReceipt, err := fileCreateTx.GetReceipt(client)
if err != nil {
    println(err.Error(), ": error getting file create transaction
receipt")
    return
}

//Get the bytecode file ID
byteCodeFileID := fileTxReceipt.FileID

fmt.Printf("The contract bytecode file ID: %v\n", byteCodeFileID)

// Deploy the contract instance
contractTransactionID, err := hedera.NewContractCreateTransaction().
    //The max gas for the transaction
    SetGas(20000000).
    //The contract bytecode file ID
    SetBytecodeFileID(byteCodeFileID).
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating contract")
    return
}

//Get the contract receipt
contractReceipt, err := contractTransactionID.GetReceipt(client)

//Get the contract contract ID
contractId := contractReceipt.ContractID

//Log the contract ID
fmt.Printf("The contract ID %v\n", contractId)

```

```

//Associate an account with a token
associateTx, err := hedera.NewTokenAssociateTransaction().
SetAccountID(accountIdTest).
SetTokenIDs(tokenId).
FreezeWith(client)

<strong>    if err != nil {
</strong>        panic(err)
    }

    //Sign with the private key of the account that is being associated to a
    token, submit the transaction to a Hedera network
    associateTxResponse, err :=
associateTx.Sign(privateKeyTest).Execute(client)

    if err != nil {
        panic(err)
    }

    //Get the receipt
    associateTxReceipt, err := associateTxResponse.GetReceipt(client)

    if err != nil {
        panic(err)
    }

    //Get transaction status
    txStatus := associateTxReceipt.Status

    fmt.Printf("The associate transaction status %v\n", txStatus)

    //Create the transaction
    transaction := hedera.NewAccountAllowanceApproveTransaction().
    ApproveHbarAllowance(ownerAccount, spenderAccountId,
Hbar.fromTinybars(500))
    FreezeWith(client)

    if err != nil {
        panic(err)
    }

    //Sign the transaction with the owner account private key
    txResponse, err := transaction.Sign(ownerAccountKey).Execute(client)

    //Request the receipt of the transaction
    receipt, err := txResponse.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    //Get the transaction consensus status
    transactionStatus := receipt.Status

    println("The transaction consensus status is ", transactionStatus)

    //Transfer the token
    transferTx := hedera.NewContractExecuteTransaction().
    //The contract ID
    SetContractID(contractId).
    //The max gas
    SetGas(20000000).
    //The contract function to call and parameters
    SetFunction("tokenTransfer", contractParamsAmount)

```

```

        //Sign with treasury key to authorize the transfer from the treasury
account
        signTx, err := transferTx.Sign(treasuryKey).Execute(client)

        if err != nil {
            println(err.Error(), ": error executing contract")
            return
        }
        //Get the receipt
        transferTxReceipt, err := signTx.GetReceipt(client)

        if err != nil {
            println(err.Error(), ": error getting receipt")
            return
        }

        //Get transaction status
        transferTxStatus := transferTxReceipt.Status

        fmt.Printf("The transfer transaction status %v\n", transferTxStatus)

        //Verify the transfer by checking the balance
        transferAccountBalance, err := hedera.NewAccountBalanceQuery().
            SetAccountID(accountIdTest).
            Execute(client)

        if err != nil {
            println(err.Error(), ": error getting balance")
            return
        }

        //Log the account token balance
        fmt.Printf("The account token balance %v\n",
transferAccountBalance.Tokens)
    }
}
</code></pre>

```

</details>

Additional Resources

🔗 Have a question? Ask on StackOverflow

🔗 Feel free to reach out in Discord!

| <p>Writer: Simi, Sr. Software Manager</p> <p> GitHub LinkedIn </p> | | | |
|---|--|--|--|
| <p>Editor: Krystal, Technical Writer</p> <p> GitHub Twitter </p> | | | |
| <p>Editor: Lucía, Developer (Hashgraph Association)</p> <p> GitHub Twitter </p> | | | |

```
td><td><a
href="https://github.com/luciamunozdev">https://github.com/luciamunozdev</a></
td></tr></tbody></table>
```

deploy-a-smart-contract-using-hardhat-hedera-json-rpc-relay.md:

description: >-

A step-by-step guide on deploying a smart contract to Hedera testnet and Hedera Local Node using Hardhat.

Deploy a Smart Contract Using Hardhat and Hedera JSON-RPC Relay

In this tutorial, you will walk through the step-by-step guide on deploying smart contracts using Hardhat and Hedera JSON-RPC Relay. Hardhat is a development environment for Ethereum. It consists of different components for editing, compiling, debugging, and deploying smart contracts and dApps, all working together to create a complete development environment.

The Hedera JSON-RPC Relay is an implementation of Ethereum JSON-RPC APIs for Hedera and utilizes both Hedera Consensus Nodes and Mirror Nodes to support RPC queries defined in the JSON-RPC Specification. The Hedera Local Node project enables developers to establish their own local network for development and testing. The local network comprises the consensus node, mirror node, JSON-RPC relay, and other Hedera products, and can be set up using the CLI tool and Docker. This setup allows you to seamlessly build and deploy smart contracts from your local environment.

By the end of this tutorial, you'll be equipped to deploy smart contracts on the Hedera Testnet or your local Hedera node, leveraging Hardhat's tools for testing, compiling, and deploying.

Prerequisites

Basic understanding of smart contracts.
Basic understanding of Node.js and JavaScript.
Basic understanding of Hardhat Ethereum Development Tool.

Table of Contents

1. Project Setup
2. Project Configuration
 1. Environment Variables
 2. Hardhat Config File
3. Compile Smart Contract
4. Deploy Smart Contract
 1. Next Steps
5. Additional Resources

Step 1: Set Up Project

To simplify the setup process, you can clone a boilerplate Hardhat example project from the hedera-hardhat-example-project repository. Open a terminal window and navigate to your preferred directory where your Hardhat project will be stored.

Run the following command to clone the repo, change into the directory, and install dependencies:

```
bash
git clone https://github.com/hashgraph/hedera-hardhat-example-project.git
cd hedera-hardhat-example-project
npm install
```

Open the project in Visual Studio Code or your IDE of choice. The project structure of the repo you just cloned should look like this:

```
hedera-hardhat-example-project
├── nodemodules
├── contracts
├── scripts
├── test
├── .env.example
├── .gitignore
├── hardhat.config.js
├── package-lock.json
├── package.json
└── README.md
```

Let's review the Hardhat project folders/content. For more information regarding Hardhat projects, check out the Hardhat docs. If you do not need to review the project contents, you can skip this optional step.

<details>

<summary>contracts/</summary>

The contracts/ folder contains the source file for the Greeter smart contract.\

Let's review the Greeter.sol contract in the hedera-example-hardhat-project/contracts folder. At the top of the file, the SPDX-License-Identifier defines the license, in this case, the MIT license. The pragma solidity ^0.8.9; line specifies the Solidity compiler version to use. These two lines are crucial for proper licensing and compatibility.

```
{% code title="Greeter.sol" %}
solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
```

```
contract Greeter {
    string private greeting;

    event GreetingSet(string greeting);

    //This constructor assigns the initial greeting and emit GreetingSet event
    constructor(string memory greeting) {
        greeting = greeting;

        emit GreetingSet(greeting);
    }

    //This function returns the current value stored in the greeting variable
    function greet() public view returns (string memory) {
        return greeting;
    }
}
```

```

    //This function sets the new greeting msg from the one passed down as
parameter and emit event
    function setGreeting(string memory greeting) public {
        greeting = greeting;

        emit GreetingSet(greeting);
    }
}

```

{% endcode %}

NOTE: The pragma solidity line must match the version of Solidity defined in the module exports of your hardhat.config.js file.

</details>

<details>

<summary>scripts/</summary>

The scripts/ folder contains test scripts for locally testing a smart contract before deploying it. Please read the comments to help you understand the code and its purpose.

contractCall.js

Calls the setGreeting function from the Greeter contract and sets the greeter message to "Greeter."

{% code title="contractCall.js" %}

```

javascript
const { ethers } = require('hardhat');

```

```

//This function accepts two parameters - address and msg
//Retrieves the contract from the address and set new greeting
module.exports = async (address, msg) => {

```

```

    //Assign the first signer, which comes from the first privateKey from our
configuration in hardhat.config.js, to a wallet variable.
    const wallet = (await ethers.getSigners())[0];

```

```

    //Assign the greeter contract object in a variable, this is used for already
deployed contract, which we have the address for. ethers.getContractAt accepts:

```

```

    //name of contract as first parameter
    //address of our contract
    //wallet/signer used for signing the contract calls/transactions with this
contract

```

```

    const greeter = await ethers.getContractAt('Greeter', address, wallet);

```

```

    //using the greeter object(which is our contract) we can call functions from
the contract. In this case we call setGreeting with our new msg

```

```

    const updateTx = await greeter.setGreeting(msg);

```

```

    console.log(Updated call result: ${msg});

```

```

    return updateTx;
};

```

{% endcode %}

contractViewCall.js

Returns the current greeter message value stored with the Greeter contract.

```
{% code title="contractViewCall.js" %}
javascript
const { ethers } = require("hardhat");

module.exports = async (address) => {
  //Assign the first signer, which comes from the first privateKey from our
  configuration in hardhat.config.js, to a wallet variable.
  const wallet = (await ethers.getSigners())[0];

  //Assign the greeter contract object in a variable, this is used for already
  deployed contract, which we have the address for. ethers.getContractAt accepts:
  //name of contract as first parameter
  //address of our contract
  //wallet/signer used for signing the contract calls/transactions with this
  contract
  const greeter = await hre.ethers.getContractAt("Greeter", address, wallet);
  //using the greeter object(which is our contract) we can call functions from
  the contract. In this case we call greet which returns our greeting msg
  const callRes = await greeter.greet();

  console.log(Contract call result: ${callRes});

  return callRes;
};

{% endcode %}
```

deployContract.js

Deploys the Greeter contract and returns the contract public address.

```
javascript
const { ethers } = require("hardhat");

module.exports = async () => {
  //Assign the first signer, which comes from the first privateKey from our
  configuration in hardhat.config.js, to a wallet variable.
  let wallet = (await ethers.getSigners())[0];

  //Initialize a contract factory object
  //name of contract as first parameter
  //wallet/signer used for signing the contract calls/transactions with this
  contract
  const Greeter = await ethers.getContractFactory("Greeter", wallet);
  //Using already intilized contract facotry object with our contract, we can
  invoke deploy function to deploy the contract.
  //Accepts constructor parameters from our contract
  const greeter = await Greeter.deploy("initialmsg");
  //We use wait to recieve the transaction (deployment) receipt, which contains
  contractAddress
  const contractAddress = (await greeter.deployTransaction.wait())
    .contractAddress;

  console.log(Greeter deployed to: ${contractAddress});

  return contractAddress;
};
```

showBalance.js

Returns the balance of the specified wallet address (account) in tinybars. Tinybars are the unit in which Hedera accounts hold HBAR balances.

```
{% code title="showBalance.js" %}
javascript
const { ethers } = require("hardhat");

module.exports = async () => {
  //Assign the first signer, which comes from the first privateKey from our
  configuration in hardhat.config.js, to a wallet variable.
  const wallet = (await ethers.getSigners())[0];
  //Wallet object (which is essentially signer object) has some built in
  functionality like getBalance, getAddress and more
  const balance = (await wallet.getBalance()).toString();
  console.log(The address ${wallet.address} has ${balance} weibars);

  return balance;
};

{% endcode %}
```

</details>

<details>

<summary>test/</summary>

The test/ folder contains the test files for the project.\

\

The rpc.js file is located in this folder in the hedera-example-hardhat-project project and references the Hardhat tasks that are defined in the hardhat.config file. When the command npx hardhat test is run, the program executes the rpc.js file.

```
{% code title="rpc.js" %}
javascript
const hre = require("hardhat");
const { expect } = require("chai");

describe("RPC", function () {
  let contractAddress;
  let signers;

  before(async function () {
    signers = await hre.ethers.getSigners();
  });

  it("should be able to get the account balance", async function () {
    const balance = await hre.run("show-balance");
    expect(Number(balance)).to.be.greaterThan(0);
  });

  it("should be able to deploy a contract", async function () {
    contractAddress = await hre.run("deploy-contract");
    expect(contractAddress).to.not.be.null;
  });

  it("should be able to make a contract view call", async function () {
    const res = await hre.run("contract-view-call", { contractAddress });
    expect(res).to.be.equal("initialmsg");
  });

  it("should be able to make a contract call", async function () {
    const msg = "updatedmsg";
    await hre.run("contract-call", { contractAddress, msg });
    const res = await hre.run("contract-view-call", { contractAddress });
    expect(res).to.be.equal(msg);
  });
});
```



```

    });
  });
{% endcode %}
</details>
<details>
<summary>.env.example</summary>

```

A file that stores your environment variables like your accounts, private keys, and references to Hedera network. Details of this file are available in Step 2 of this tutorial.

```

</details>
<details>
<summary>hardhat.config.js</summary>

```

The Hardhat configuration file. This file includes information about the Hedera network RPC URLs, accounts, and tasks defined. Details of this file are available in Step 2 of this tutorial.

```

</details>

```

Step 2: Configure Project

In this step, you will update and configure your environment variables and Hardhat configuration files that define tasks, store account private keys, and RPC endpoint URLs. First, rename the .env.exmaple file to .env.

Environment Variables

The .env file securely stores environment variables, such as your Hedera network endpoints and private keys, which are then imported into the hardhat.config.js file. This helps protect sensitive information like your private keys and API secrets, but it's still best practice to add .env to .gitignore file to prevent you from committing and pushing your credentials to GitHub. Go to the tab corresponding to your deployment path and follow the steps to set up your environment variables.

```

{% tabs %}
{% tab title="local node" %}
Prerequisite: A Hedera Local Node set up and running (setup tutorial).

```

Hedera Local Node environment variables

The variables are predefined for the purposes of this tutorial.

```

{% code title=".env" %}
bash
Your Hedera Local Node ECDSA account alias private key
LOCALNODEOPERATORPRIVATEKEY=0x105d050185ccb907fba04dd92d8de9e32c18305e097ab41dad
da21489a211524
Your Hedera Local Node JSON-RPC endpoint URL
LOCALNODEENDPOINT='http://localhost:7546/'
{% endcode %}

```

Variables explained

LOCALNODEOPERATORPRIVATEKEY: This is your Alias ECDSA hex-encoded private key for your Hedera Local Node. Replace the example value with your actual private key. Once you set up your local node and run the command to start, the accounts list for alias ECDSA private keys will be generated and returned to your console (see screenshot below). Replace the example value with your actual private key.

<figure><figcaption></figcaption></figure>

LOCALNODEENDPOINT: This is the URL endpoint for your Hedera Local Node's JSON-RPC Relay. Typically, this would be your localhost followed by the port number (http://localhost:7546/).

<figure><figcaption></figcaption></figure>
{% endtab %}

{% tab title="testnet" %}

Prerequisite: A Hedera testnet account from the Hedera Developer Portal.

Hedera Testnet environment variables

{% code title=".env" %}

bash

Your testnet account ECDSA hex-encoded private key

TESTNETOPERATORPRIVATEKEY=0xb46751179bc8aa9e129d34463e46cd924055112eb30b31637b5081b56ad96129

Your testnet JSON-RPC Relay endpoint URL

TESTNETENDPOINT='https://testnet.hashio.io/api'

{% endcode %}

Variables explained

TESTNETOPERATORPRIVATEKEY: This is your ECDSA hex-encoded private key for the Hedera Testnet. Replace the example value with your actual private key.

<figure><figcaption></figcaption></figure>

TESTNETENDPOINT: This is the URL endpoint for the Hedera Testnet's JSON-RPC Relay. Replace the example URL with the one you're using.

For this tutorial, we'll use Hashio, an instance of the Hedera JSON-RPC relay hosted by Swirlds Labs. You can use any JSON-RPC instance the community supports.

{% content-ref url="../../more-tutorials/json-rpc-connections/hashio.md" %}

hashio.md

{% endcontent-ref %}

{% endtab %}

{% endtabs %}

Configuring these environment variables enables your Hardhat project to interact with the Hedera network or your local node. Let's review the Hardhat configuration file, where these environment variables are loaded into.

Hardhat Configuration File

The Hardhat config (hardhat.config.js) file serves as the central configuration file for your Hardhat project. This file is crucial for specifying various

settings, including Hardhat tasks, network configurations, compiler options, and testing settings. Let's review the configuration settings.

Required Packages and Mocha Settings

These first lines import the required Hardhat plugins and the dotenv module that loads environment variables from the .env file. This will allow you to keep your private keys secure while using them in your dApp and will keep you from committing these to GitHub.

```
{% code title="hardhat.config.js" %}
javascript
require("@nomicfoundation/hardhat-toolbox");
require("@nomicfoundation/hardhat-chai-matchers");
require("@nomiclabs/hardhat-ethers");
require("dotenv").config(); // Import dotenv library to access the .env file

{% endcode %}
```

Hardhat Tasks

These lines define tasks that are accessed and executed from the test/ or scripts/ folders.

```
{% code title="hardhat.config.js" %}
javascript
//define hardhat task here, which can be accessed in our test file (test/rpc.js)
by using hre.run('taskName')
task("show-balance", async () => {
  const showBalance = require("../scripts/showBalance");
  return showBalance();
});

task("deploy-contract", async () => {
  const deployContract = require("../scripts/deployContract");
  return deployContract();
});

task("contract-view-call", async (taskArgs) => {
  const contractViewCall = require("../scripts/contractViewCall");
  return contractViewCall(taskArgs.contractAddress);
});

task("contract-call", async (taskArgs) => {
  const contractCall = require("../scripts/contractCall");
  return contractCall(taskArgs.contractAddress, taskArgs.msg);
});

{% endcode %}
```

Solidity Compiler Settings

Here, the Solidity compiler version is set to "0.8.9". The optimizer is enabled with 500 runs to improve the contract's efficiency.

```
{% code title="hardhat.config.js" %}
javascript
/ @type import('hardhat/config').HardhatUserConfig /
module.exports = {
  mocha: {
    timeout: 3600000,
  },
  solidity: {
    version: "0.8.9",
```

```

settings: {
  optimizer: {
    enabled: true,
    runs: 500,
  },
},
},
},

```

{% endcode %}

Network Configurations

The networks object is essential for defining the Hedera networks your Hardhat project will connect to. Additionally, the defaultNetwork key specifies the network Hardhat will default to if none is specified at deployment.

```

{% tabs %}
{% tab title="local node" %}
{% code title=".env" %}
javascript
// Specifies which network configuration will be used by default when you run
Hardhat commands.
defaultNetwork: "local",
networks: {
  // Defines the configuration settings for connecting to Hedera local node
  local: {
    // Specifies URL endpoint for Hedera local node pulled from the .env file
    url: process.env.LOCALNODEENDPOINT,
    // Your local node operator private key pulled from the .env file
    accounts: [process.env.LOCALNODEOPERATORPRIVATEKEY],
  },
},

```

{% endcode %}

{% endtab %}

```

{% tab title="testnet" %}
{% code title=".env" %}
javascript
// Specifies which network configuration will be used by default when you run
Hardhat commands.
defaultNetwork: "testnet",
networks: {
  // Defines the configuration settings for connecting to Hedera testnet
  testnet: {
    // Specifies URL endpoint for Hedera testnet pulled from the .env file
    url: process.env.TESTNETENDPOINT,
    // Your ECDSA testnet account private key pulled from the .env file
    accounts: [process.env.TESTNETOPERATORPRIVATEKEY],
  },
},

```

{% endcode %}

{% endtab %}

{% endtabs %}

Key/value breakdown:

defaultNetwork: This property specifies which network configuration will be used by default when you run Hardhat commands.

networks: This property contains configurations for different networks you might connect to.

url: This specifies the URL endpoint for the network. The value is pulled from the .env file where the environment variables are defined.


accounts: This lists the private keys for the accounts you'll use when connecting to the network. The value is pulled from the .env file where the environment variables are defined.

Step 3: Compile Contract

Now that your project is configured compile your contract. Run the following command in the hedera-hardhat-example-project terminal:

```
bash
npx hardhat compile
```

<details>

<summary>Console check  </summary>

```
bash
Compiling...
Compiled 1 contract successfully
```

</details>

The compiled artifacts will be saved in the artifacts/ directory by default, or whatever your configured artifacts path is. The metadata file generated in this directory will be used for the smart contract verification process in a later step.


<figure><figcaption></figcaption></figure>

After the initial compilation, if you don't modify any files, nothing will be compiled when you run the compile command. To force a compilation you can use the --force flag or run npx hardhat clean to clear the cache and delete the artifacts to recompile.

Step 4: Test and Deploy Contract

Once your contract is compiled successfully, deploy the Greeter.sol smart contract. There are additional steps required if you're deploying to your local node:

<details>

<summary> local node additional steps</summary>

Before you deploy your contract, let's ensure you have all the necessary tools open and running to avoid any issues.

Have two terminals open. One for each of these two project directories:

hedera-local-node

hedera-hardhat-example-project

Have Docker open and start your local node.

In the hedera-local-node terminal, start your local node by running hedera start -d.

Note: If you have not set up your Hedera Local Node, you can do so by following this tutorial and returning to this step once you complete the setup.

</details>

Test

Test your contract before deploying it. In the `hedera-hardhat-example-project` terminal, run the following command to compile and test your contract:

```
bash
npx hardhat test
```

Tests should pass with "**4 passing**" returned to the console. Otherwise, an error message will appear indicating the issue.

<details>

```
<summary>console check ☒
```

```
shell
RPC
The address 0xe261e26aECcE52b3788Fac9625896FFbc6bb4424 has
999999999999999999991611392 weibars
    ✓ should be able to get the account balance (1127ms)
Greeter deployed to: 0xEc3D74D360a53Fe7104Be6aB4e25e27a90bF6aE4
    ✓ should be able to deploy a contract (11810ms)
Contract call result: initialmsg
    ✓ should be able to make a contract view call (265ms)
Updated call result: updatedmsg
Contract call result: updatedmsg
    ✓ should be able to make a contract call (4068ms)
```

4 passing (34s)

</details>

Deploy

In the same terminal, run the following command to deploy your contract to the default network specified in your config file:

```
bash
npx hardhat deploy-contract
```

Alternatively, you can target any network configured in your Hardhat config file. For testnet:

```
bash
npx hardhat run --network testnet scripts/deployContract.js
```

<details>

```
<summary>console check ☒
```

```
bash
Greeter deployed to: 0x157B93c04a294AbD88cF608672059814b3ea38aE
```

</details>

Next Steps

```
{% tabs %}
{% tab title="local node" %}
Stop Local Node
```

Stop your local node and remove Docker containers by running `hedera stop` or `docker compose down` in your `hedera-local-node` terminal. Reference the [Stop Your Local Node](#) section of the local node setup tutorial.

Deploy on Hedera Testnet

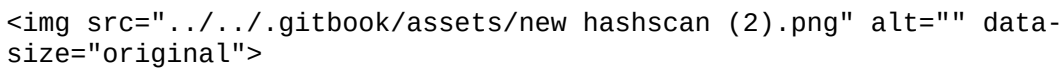
If you're up for it, follow the steps to deploy on the Hedera testnet and verify your contract.

```
{% endtab %}
```

```
{% tab title="testnet" %}
View Contract on HashScan Network Explorer
```

You can view the contract you deployed by searching the smart contract public address in a supported Hedera Network Explorer. For this example, we will use the HashScan Network Explorer. Copy and paste your deployed Greeter.sol public contract address into the HashScan search bar.

The Network Explorer will return the information about the contract created and deployed to the Hedera Testnet. The "EVM Address" field is the public address of the contract that was returned to you in your terminal. The terminal returned the public address with the "0x" hex encoding appended to the public address. You will also notice a contract ID in `0.0.contractNumber (0.0.3478001)` format. This is the contract ID used to reference the contract entity in the Hedera Network.

 ``

Verify Contract

Additionally, you can verify your contract using the HashScan verification feature (beta). Follow these steps to learn how.

Note: At the top of the explorer page, remember to switch the network to TESTNET before you search for the contract.

```
{% endtab %}
{% endtabs %}
```

Congratulations! 🎉 You have successfully learned how to deploy a smart contract using Hardhat and Hedera JSON-RPC Relay. Feel free to reach out in [Discord](#)!

Additional Resources

- [🔗 Project Repository](#)
- [🔗 Hedera Local Node Repository](#)
- [🔗 Hedera JSON-RPC Relay Repository](#)
- [🔗 Hedera Local Node Setup Tutorial](#)
- [🔗 Hardhat Documentation](#)

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
```

<p>Writer: Krystal, Technical Writer</p> <p> GitHub Hashnode </p>	<p> https://github.com/theekrystallee </p>
<p>Editor: Simi, Sr. Software Manager</p> <p> GitHub LinkedIn </p>	<p> https://www.linkedin.com/in/shunjan/ </p>
<p>Editor: Nana, Sr Software Manager</p> <p> GitHub LinkedIn </p>	<p> https://www.linkedin.com/in/nconduah/ </p>
<p>Editor: Georgi, Sr Software Dev (LimeChain)</p> <p> GitHub LinkedIn </p>	<p> https://github.com/georgi-l95 </p>

deploy-a-smart-contract-using-remix.md:

```

---
description: >-
  A step-by-step tutorial on how to create and deploy a smart contract on the
  Hedera network using Remix IDE.
---
```

Deploy a Smart Contract Using Remix

Introduction to Remix

Remix IDE is an open-source tool for developing smart contracts in Solidity for the Ethereum network. It offers built-in compiling, debugging, and deploying features, streamlining the development process. This tutorial will leverage Remix IDE to create and deploy a simple smart contract on the Hedera network.

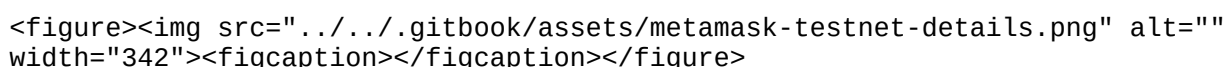
Prerequisites

- Web browser with access to Remix IDE.
- Create a Hedera ECDSA testnet account.
- Download the MetaMask wallet browser extension.

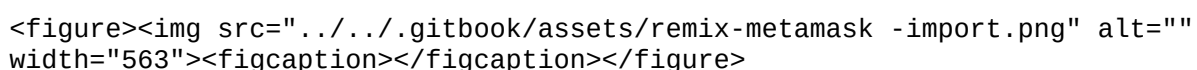
Add Hedera Testnet to your MetaMask


Before deploying a smart contract to the Hedera network using Remix, add Hedera Testnet as a custom network to MetaMask.

1. Open the MetaMask wallet extension and navigate to Settings > Networks > Add a Network > Add a network manually then add the Hedera Testnet details:



2. Switch the network to Hedera Testnet and add your account by importing your hex-encoded ECDSA private key to MetaMask.



 Here is a more comprehensive guide on importing a Hedera account into

MetaMask.

Create a Smart Contract

Open your web browser and navigate to Remix IDE. Click on the file icon in the File Explorer tab to create a new file and name it HelloHedera.sol .

<figure><figcaption></figcaption></figure>

Copy and paste this sample contract to the new file you created:

```
solidity
//SPDX-License-Identifier: MIT

pragma solidity 0.8.22;

contract SampleContract {
    string public myString = "Hello Hedera";

    function updateString(string memory newString) public {
        myString = newString;
    }
}
```

Compile the Contract

Navigate to the Solidity Compiler tab in the left sidebar and check that your compiler version is within the versions specified in the pragma solidity statement. Then, compile your HelloHedera.sol contract.

<figure><figcaption></figcaption></figure>

When a compilation for a Solidity file succeeds, Remix creates three JSON files for each compiled contract. Files can be seen in the File Explorers plugin as:

1. artifacts/<contractName>.json: contains the link to the libraries, the bytecode, the deployed bytecode, the gas estimation, the method identifiers, and the ABI. It is used for linking a library address to the file.
2. artifacts/<contractName>metadata.json: contains the metadata from the output of Solidity compilation.
3. artifacts/build-info/<dynamicHash>.json: contains info about solc compiler version, compiler input and output. This file is generated similar to the files generated through Hardhat compilation. You can also try Hardhat compilation from Remix.

<figure><figcaption></figcaption></figure>

Please note that to generate these artifact files, the Generate contract metadata box in the General settings section of the Settings module needs to be checked. By default, it is checked.

Deploy to Hedera Testnet

Go to the Deploy & Run Transactions tab and select Injected Provider - MetaMask

as the environment. A window will pop up if you're not signed into your MetaMask account. Sign in and make sure you're on Hedera Testnet and verify that the network is configured properly to Custom (296) network.

<figure><figcaption></figcaption></figure>

Once you click Deploy in the Deploy & Run Transactions tab, hit Confirm in the MetaMask notification window to approve and pay for the contract deployment transaction.

<figure><figcaption></figcaption></figure>

Interact with the Smart Contract on Hedera

Once the transaction is successful, you can interact with the smart contract through Remix. Select the dropdown on the newly deployed contract at the bottom of the left panel to view the contract's functions under Deployed Contracts. Write a new message to the updateString function using the input and confirm the write transaction in the MetaMask window to pay.

<figure><figcaption></figcaption></figure>

View Contract Details

Copy the contract address from the Deployed Contracts window.

<figure><figcaption></figcaption></figure>

Navigate to the HashScan network explorer and use the contract address to search for your contract to view the details.

<figure><figcaption></figcaption></figure>

Next Steps: Verify Your Smart Contract

If you're up for it, you can verify your deployed contract using the HashScan Smart Contract Verifier tool. Learn how:

{% content-ref url="how-to-verify-a-smart-contract-on-hashscan.md" %}
how-to-verify-a-smart-contract-on-hashscan.md
{% endcontent-ref %}

Congratulations! 🎉 You have successfully deployed a smart contract on the Hedera network using Remix IDE. Feel free to reach out in Discord if you have any questions!

Additional Resources

- ➞ [Remix IDE Documentation](#)
- ➞ [HashScan Network Explorer](#)
- ➞ [Deploy Leveraging EVM Dev Tools](#)

<p>Writer: Krystal, Technical Writer</p> <p> GitHub Hashnode </p>	https://hashnode.com/@theekrystallee
<p>Editor: Abi, DevRel Engineer</p> <p> GitHub LinkedIn </p>	https://www.linkedin.com/in/a-ridley/
<p>Editor: Logan, Software Engineering Intern</p> <p> GitHub LinkedIn </p>	https://github.com/quiet-node

deploy-a-subgraph-using-the-graph-and-json-rpc.md:

Deploy a Subgraph Using The Graph and Hedera JSON-RPC Relay

In this tutorial, you'll learn how to create and deploy a subgraph using The Graph protocol. By indexing specific network data using user-defined data structures called "subgraphs," developers can easily query the indexed data through a GraphQL API, creating robust backends for dApps. Subgraphs simplify the process of obtaining blockchain/network data for developers building dApps. This approach removes the complexities of interacting directly with the network, allowing developers to focus on building. Although Hedera supports subgraphs, its hosted service is currently unavailable, so we'll need to set up and run a local graph node to deploy our subgraph.

By the end of this tutorial, you'll be able to configure a mirror node, query data from your subgraph using the GraphQL API, and integrate it into your dApp. You'll also have a better understanding of how to define custom data schemas, indexing rules, and queries for your subgraph, allowing you to tailor it to your specific use case.

{% hint style="info" %}

Note: While it is possible to present and interact with HTS tokens in a similar manner as ERC-20/721 tokens, the network is presently unable to capture all the expected ERC-20/721 event logs. In other words, if ERC-like operations are conducted on HTS tokens, not all of them will be captured in smart contract event logging.

{% endhint %}

Prerequisites

- Basic understanding of JavaScript and NPM installed.
- Basic understanding of subgraphs and the Graph CLI installed.
- The deployed Greeter smart contract address from the Hardhat tutorial.
- The start block number of when the Greeter smart contract was first deployed.
- Docker >= v20.10.x installed and open on your machine. Run `docker -v` in your terminal to check the version you have installed.

<details>

<summary>Find start block</summary>

1. Go to HashScan explorer here.
2. Enter your public contract address or contract ID in the search bar.

3. Click on the Create Transaction ID (0.0.902@1676712828.922009885).

Note: When searching for contract addresses, there are two types with different formats - the public smart contract address (0x....) or contract ID (0.0.12345).

</details>

<details>

<summary>Graph CLI installation</summary>

Open your terminal and run the following command:

```
bash
npm install -g @graphprotocol/graph-cli
```

Test to see if it was installed correctly by running:

```
bash
graph -v
```

Note: The Graph CLI will be installed globally, so you can run the command in any directory.

</details>

Table of Contents

1. Project Setup
2. Project Configuration
3. Deploy Subgraph
4. Code Check
5. Additional Resources

Project Setup

Open a terminal window and navigate to the directory where you want your subgraph project stored. Clone the hedera-subgraph-example repo, change directories, and install dependencies:

```
bash
git clone https://github.com/hashgraph/hedera-subgraph-example.git
cd hedera-subgraph-example
npm install
```

Rename the subgraph.template.yaml file to subgraph.yaml before moving on to the next step. The subgraph project structure should look something like this:

```
subgraph-name
├──abis
│   └── IGreeter.json
├──config
│   └──testnet.json
└──
```

```

├── graph-node
│   └── docker-compose.yaml
└── src
    ├── mappings.ts
    ├── package-lock.json
    ├── package.json
    ├── README.md
    ├── schema.graphql
    └── subgraph.yaml

```

In the testnet.json file, under the config folder, replace the startBlock and Greeter fields with your start block number and contract address. The JSON file should look something like this:

```

{% code title="testnet.json" %}
json
{
  "startBlock": 1050018,
  "Greeter": "0xCc0d40EA9d2Dd16Ab5565ae91b121960d5e19e4e"
}

{% endcode %}

```

Project Configuration

In this step, you will use the Greeter contract from the Hardhat tutorial as an example subgraph, to configure four main project files: the subgraph manifest, GraphQL schema, event mappings, and Docker compose configuration. The manifest specifies which events the subgraph will listen for, while mappings map each event emitted by the smart contract into entities that can be indexed.

Subgraph Manifest

The subgraph manifest (subgraph.yaml) contains important information about your subgraph, such as its name, description, and data sources. To specify the data sources your subgraph will index, you need to define the dataSources field in the manifest. It's also recommended to add the start block number to the startBlock property to reduce the indexing time. Here's a guide on how to find the start block number.

1. Add your deployed Greeter public smart contract address to the address property.
2. Add your start block number of the deployed contract in the startBlock property.

```

{% code title="subgraph.yaml" %}
yaml
dataSources:
  - kind: ethereum/contract
    name: Greeter
    network: testnet
    source:
      Step 1
      address: "0xCc0d40EA9d2Dd16Ab5565ae91b121960d5e19e4e"
      abi: IGreeter
      Step 2
      startBlock: 1050018

{% endcode %}

```

The `eventHandlers` field specifies how each mapping connects to various event triggers. Whenever an event defined in this section is emitted from your contract, the corresponding mapping function designated as the handler will be executed.

```
{% code title="subgraph.yaml" %}
yaml
  eventHandlers:
    - event: GreetingSet(string)
      handler: handleGreetingSet
    file: ./src/mappings.ts
```

```
{% endcode %}
```

GraphQL Schema

The GraphQL schema (`schema.graphql`) defines the structure of the data you want to index in your subgraph. You will need to specify the entity properties that you want to index. For this example, the schema defines a GraphQL entity type called "Greeting" with two entity fields: `id` and `currentGreeting`.

```
{% code title="schema.graphql" %}
graphql
type Greeting @entity {
  id: ID!
  currentGreeting: String!
}
```

```
{% endcode %}
```

Event Mappings

The `mappings.ts` file maps events emitted by your smart contract into entities that can be indexed by a subgraph. It uses AssemblyScript to connect the events to the data schema. AssemblyScript types for entities and events can be generated in the terminal (by running the `codegen` command) and imported into the mappings file. This allows easy access to the event object's properties in the code editor.

```
{% code title="mappings.ts" %}
typescript
import { GreetingSet } from '../generated/Greeter/IGreeter';
import { Greeting } from "../generated/schema";

export function handleGreetingSet(event: GreetingSet): void {
  // Entities can be loaded from the store using a string ID; this ID
  // needs to be unique across all entities of the same type
  let entity = Greeting.load(event.transaction.hash.toHexString());

  // Entities only exist after they have been saved to the store;
  // null checks allow to create entities on demand
  if (!entity) {
    entity = new Greeting(event.transaction.hash.toHex());
  }

  // Entity fields can be set based on event parameters
  entity.currentGreeting = event.params.greeting;

  // Entities can be written to the store with .save()
  entity.save();
}
```

```
{% endcode %}
```

Graph Node Configuration

To connect a local graph node to a remote network, such as testnet, mainnet, or previewnet, use a docker-compose setup. The API endpoint that connects the graph node to the network is specified within the environment object of the docker-compose.yaml file here. Add the API endpoint URL in the ethereum field in the environment object. For this tutorial, we will use the Hashio Testnet instance of the Hedera JSON-RPC relay, but any JSON-RPC provider supported by the community can be used.

This is what the ethereum field should look like after you enter your API endpoint URL:

```
yaml
ethereum: 'testnet:https://testnet.hashio.io/api'
```

Note: For more info on how to set up an indexer, check out The Graph\ docs and the official graph-node GitHub repository. For a full subgraph project example, check out this repo.

Deploy Subgraph


In this step, you will create the subgraph and deploy it to your local graph node. If everything runs without errors, your terminal should resemble the console check at the end of each subsection.

1. Start Graph Node

To start your local graph node, have the Docker engine running before executing the below command in your project directory:

```
bash
npm run graph-node
```

<details>

<summary>console check  </summary>

The first time you run the command:

```
bash
Creating graph-nodepostgres1 ... done
Creating graph-nodeipfs1      ... done
Creating graph-nodegraph-node1 ... done
```

What your console will return if you run the command more than once:

```
bash
[+] Running 3/0
  :: Container graph-node-postgres-1    Running    0.0s
  :: Container graph-node-ipfs-1        Running    0.0s
  :: Container graph-node-graph-node-1  Running    0.0s
```

</details>

2. Generate Types

In the same directory, run the following command to generate AssemblyScript types for entities and events:

```
bash
graph codegen
```

<details>

<summary>console check ☒

```
  Skip migration: Bump manifest specVersion from 0.0.2 to 0.0.4
  ✓ Apply migrations
  ✓ Load subgraph from subgraph.yaml
    Load contract ABI from abis/IGreeter.json
  ✓ Load contract ABIs
    Generate types for contract ABI: IGreeter (abis/IGreeter.json)
    Write types to generated/Greeter/IGreeter.ts
  ✓ Generate types for contract ABIs
  ✓ Generate types for data source templates
  ✓ Load data source template ABIs
  ✓ Generate types for data source template ABIs
  ✓ Load GraphQL schema from schema.graphql
    Write types to generated/schema.ts
  ✓ Generate types for GraphQL schema
```

Types generated successfully

</details>

You should have a new folder named generated in your project directory. This is what your updated subgraph project structure should look like:

```
subgraph-name
├── abis
│   └── IGreeter.json
├── config
│   └── testnet.json
├── generated
│   ├── Greeter
│   │   └── IGreeter.ts
│   └── schema.ts
├── graph-node
│   └── docker-compose.yaml
├── src
│   ├── mappings.ts
│   ├── package-lock.json
│   ├── package.json
│   ├── README.md
│   ├── schema.graphql
│   └── subgraph.yaml
```

3. Create and Deploy

To create and deploy your subgraph to your local graph node, run:

```
bash
// create the subgraph
```



```
npm run create-local
```

<details>

<summary>console check ☒

```
bash
```

```
> hedera-subgraph-repo-example@1.0.0 create-local
> graph create --node http://localhost:8020/ Greeter
```

Created subgraph: Greeter

</details>

```
bash
```

```
// deploy the subgraph
npm run deploy-local
```

When you run the deploy-local command, your console will prompt you to provide a Version Label. Enter any version number you'd like. This is just a way to keep track of different versions of your subgraph. For instance, if you started with version v0.0.1 today, but then made some changes and wanted to deploy an upgraded version, you bump up the version number to v0.0.2.

For example: ☒ Version Label (e.g. v0.0.1) · <mark style="background-color:yellow;">v0.0.1</mark>

<details>

<summary>console check ☒

```
bash
```

```
> hedera-subgraph-repo-example@1.0.0 deploy-local
> graph deploy --node http://localhost:8020/ --ipfs http://localhost:5001
Greeter
```

- ✓ Version Label (e.g. v0.0.1) · v0.0.1
- ✓ Apply migrations
- ✓ Load subgraph from subgraph.yaml
 - Compile data source: Greeter => build/Greeter/Greeter.wasm
- ✓ Compile subgraph
 - Copy schema file build/schema.graphql
 - Write subgraph file build/Greeter/abis/IGreeter.json
 - Write subgraph manifest build/subgraph.yaml
- ✓ Write compiled subgraph to build/
 - Add file to IPFS build/schema.graphql
 - .. QmVtZMzbjU6QHEFfrCJ5NhbP5vUNrukaussxXZ4Esf3qCm
 - Add file to IPFS build/Greeter/abis/IGreeter.json
 - .. QmZQbrdhaR2p2EZR6raiLbpgX5hJKW4S5cDgy1VvHKmjth
 - Add file to IPFS build/Greeter/Greeter.wasm
 - .. QmYz3qFZ4KHihXhgbTKFxbCNvE9Serhq8yvGuJPK12K5qf
- ✓ Upload subgraph to IPFS

Build completed: QmbGuuuqtEEqFxjdwSdhiKKpb4GCzqbh3oASAnVVEXRoVw

Deployed to http://localhost:8000/subgraphs/name/Greeter/graphql

Subgraph endpoints:

Queries (HTTP): http://localhost:8000/subgraphs/name/Greeter

After the subgraph is successfully deployed, open the GraphQL playground, where you can execute queries and fetch indexed data.

</details>

Code Check ☒

<details>

<summary>subgraph.yaml</summary>

```
yaml
specVersion: 0.0.4
description: Graph for Greeter contracts
repository: https://github.com/hashgraph/hedera-subgraph-example
schema:
  file: ./schema.graphql
dataSources:
  - kind: ethereum/contract
    name: Greeter
    network: testnet
    source:
      address: "0xCc0d40EA9d2Dd16Ab5565ae91b121960d5e19e4e"
      abi: IGreeter
      startBlock: 1050018
    mapping:
      kind: ethereum/events
      apiVersion: 0.0.6
      language: wasm/assemblyscript
      entities:
        - Greeting
      abis:
        - name: IGreeter
          file: ./abis/IGreeter.json
      eventHandlers:
        - event: GreetingSet(string)
          handler: handleGreetingSet
      file: ./src/mappings.ts
```

</details>

<details>

<summary>docker-compose.yaml</summary>

docker-compose.yaml

```
yaml
version: '3'
services:
  graph-node:
    image: graphprotocol/graph-node:v0.27.0
    ports:
      - '8000:8000'
      - '8001:8001'
      - '8020:8020'
      - '8030:8030'
      - '8040:8040'
    dependson:
      - ipfs
      - postgres
```

```

extrahosts:
  - host.docker.internal:host-gateway
environment:
  postgreshost: postgres
  postgresuser: 'graph-node'
  postgrespass: 'let-me-in'
  postgresdb: 'graph-node'
  ipfs: 'ipfs:5001'
  ethereum:
    GRAPHLOG: info
    GRAPHETHEREUMGENESISBLOCKNUMBER: 1
ipfs:
  image: ipfs/go-ipfs:v0.10.0
  ports:
    - '5001:5001'
  volumes:
    - ./data/ipfs:/data/ipfs
postgres:
  image: postgres
  ports:
    - '5432:5432'
  command:
    [
      "postgres",
      "-csharedpreloadlibraries=pgstatstatements"
    ]
  environment:
    POSTGRESUSER: 'graph-node'
    POSTGRESPASSWORD: 'let-me-in'
    POSTGRESDB: 'graph-node'
    PGDATA: "/data/postgres"
  volumes:
    - ./data/postgres:/var/lib/postgresql/data

```

</details>

Congratulations! You've successfully deployed a subgraph to your local graph node!

Once the node finishes indexing, you can access the GraphQL API at:
<http://localhost:8000/subgraphs/name/Greeter>

Follow the steps below to execute the query and fetch the indexed data from the subgraph's entities:

1. Enter the following GraphQL query into the left column of the playground (see Step 1 in the screenshot below):

```

graphql
{
  greetings {
    id
    currentGreeting
  }
}

```

2. Execute the query by clicking on the play button at the top of the playground (see Step 2 in the screenshot below).

3. The query returns the indexed data from the subgraph's entities on the right column of the playground (see Step 3 in the screenshot below):

```

graphql

```

```
{
  "data": {
    "greetings": [
      {
        "id":
"0xe30c4a439ffbcf4a7e9f3083ec07cc056f456770d080f2f08cc546a399d71516",
        "currentGreeting": "initialmsg"
      }
    ]
  }
}
```

<figure><figcaption><p>GraphQL Playground</p></figcaption></figure>

Congratulations! 🎉 You have successfully learned how to deploy a Subgraph using The Graph Protocol and JSON-RPC. Feel free to reach out on Discord if you have any questions!

Additional Resources

🔗 Project Repository

🔗 Subgraph Example

<p>Writer: Krystal, Technical Writer</p> <p> GitHub Hashnode </p>	https://github.com/theekrystallee/
<p>Editor: Simi, Sr. Software Manager</p> <p> GitHub LinkedIn </p>	https://www.linkedin.com/in/shunjan/
<p>Editor: Georgi, Sr Software Dev (LimeChain)</p> <p> GitHub LinkedIn </p>	https://github.com/georgi-l95/

deploy-by-leveraging-ethereum-developer-tools-on-hedera.md:

Deploy By Leveraging Ethereum Developer Tools On Hedera

Learning how to properly use new developer tools requires time and effort. Many seasoned engineers already have their own, reliable set of frameworks and libraries they frequently use. With the release of the Hedera JSON-RPC relay, Ethereum developer tools combined with ECDSA-based Hedera accounts are available for developers only. You can continue to utilize familiar Ethereum tooling to build on Hedera. This blog speaks to the support of 4 Ethereum tools and the enablement of Metamask.

Supported Ethereum Developer Tools

The most common EVM-based tools and workflows across Web3 ecosystems are built on the JSON-RPC specification. You can continue to utilize the following familiar Ethereum tooling, Web3JS, Truffle, Ethers, and Hardhat, to build on Hedera, thanks to the JSON-RPC Relay. As an Ethereum developer, your workflow

does not have to change.

```
<table data-header-hidden><thead><tr><th width="219"></th><th width="124"
align="center"></th><th width="142" align="center"></th><th width="117"
align="center"></th></thead><tbody><tr><td><br></td><td
align="center"><strong>web3js</strong></td><td
align="center"><strong>Truffle</strong></td><td
align="center"><strong>ethers</strong></td><td
align="center"><strong>Hardhat</strong></td></tr><tr><td>Transfer HBARS</td><td
align="center">✔</td><td align="center">✔</td><td align="center">✔</td><td
align="center">✔</td></tr><tr><td>Contract Deployment</td><td
align="center">✔</td><td align="center">✔</td><td align="center">✔</td><td
align="center">✔</td></tr><tr><td>Can use the contract instance after deploy
without re-initialization</td><td align="center">✔</td><td
align="center">✔</td><td align="center">△</td><td
align="center">△</td></tr><tr><td>Contract View Function Call</td><td
align="center">✔</td><td align="center">✔</td><td align="center">✔</td><td
align="center">✔</td></tr><tr><td>Contract Function Call</td><td
align="center">✔</td><td align="center">✔</td><td align="center">✔</td><td
align="center">✔</td></tr></tbody></table>
```

You can transfer HBAR, deploy contracts, and perform contract calls bringing even greater usability to the developer community.

Check out the Web3js, Truffle, and Hardhat examples on the repo. It is important to note that when working with Ethersjs and Hardhat, there is an extra step to retrieve the valid Hedera contract address. Learn more about it [here](#).

Getting Started

Before you start, creating a new ECDSA-based account with an alias is important. Currently, the JSON-RPC Relay only supports Hedera accounts with an alias set (i.e., public address) based on its ECDSA public key. You can easily do this by following the steps below:

```
javascript
// generate an ECDSA key-pair
const newPrivateKey = PrivateKey.generateECDSA();
console.log(The raw private key (use this for JSON RPC wallet import): $
{newPrivateKey.toStringRaw()});

const newPublicKey = newPrivateKey.publicKey;

// account publickey alias
const aliasAccountId = newPublicKey.toAccountId(0, 0);
console.log(The alias account id: ${aliasAccountId});

javascript
const operatorAccountId = AccountId.fromString(process.env.OPERATORID);
const operatorPrivateKey = PrivateKey.fromString(process.env.OPERATORPVKEY);

// Hbar transfers will auto-create a Hedera Account
// for long-form account Ids that do not have accounts yet
const tokenTransferTxn = async (senderAccountId, receiverAccountId, hbarAmount)
=> {
  const transferToAliasTx = new TransferTransaction()
    .addHbarTransfer(senderAccountId, new Hbar(-hbarAmount))
    .addHbarTransfer(receiverAccountId, new Hbar(hbarAmount))
    .freezeWith(client);

  const signedTx = await transferToAliasTx.sign(operatorPrivateKey);
  const txResponse = await signedTx.execute(client);
```

```

    await txResponse.getReceipt(client);
}

```

Create a function to help log your account info.

```

javascript
const logAccountInfo = async (accountId) => {
    const info = await new AccountInfoQuery()
        .setAccountId(accountId)
        .execute(client);

    console.log(The normal account ID: ${info.accountId});
    console.log(Account Balance: ${info.balance});
}

```

```

javascript
const main = async () => {
    await tokenTransferTxn(operatorAccountId, aliasAccountId, 5);
    await logAccountInfo(aliasAccountId);
}

```

```
main();
```

Console output

```

javascript
The raw private key (use this for JSON RPC wallet import):
81cd442a945d2c9f04ed5bf355a59db9e9f7553b9d4c319938eb9176085cb4c8
The alias account id:
0.0.302d300706052b8104000a03220002d47f1da5a3e086c568776d5be31165c65a135bb48951b4
ccb4f4284b025225ff4
The normal account ID: 0.0.47995491
Account Balance: 99.31142415 ¢

```

The account is officially registered with Hedera when HBAR is initially deposited to the account alias. The transaction fee to create the account is deducted from the initial hbar transfer. The remaining balance, minus the transaction fee to create the account, is the initial balance of the new account. If you want to learn more about auto account creation, read the following documentation and HIP-32.

> IMPORTANT NOTE: Private keys for Testnet are displayed here for educational purposes only. Never share your private key(s) with others, as that may result in lost funds or loss of control over your account.

Import Hedera Account into Metamask

Step 1: Go to Hashio, the Hashgraph-hosted version of the JSON-RPC Relay, and copy the Testnet URL.

Hashio provides the URLs for each Hedera environment, which allows you to interact with the respective environment nodes on Hedera the same way you would an Ethereum node.

```

{% embed url="https://www.canva.com/design/DAGMF5F94f8/3n2pCHTX8nUiN07uiQzomg/view" %}
https://www.hashgraph.com/hashio/
{% embed %}

```

Step 2: Open MetaMask and add Hedera as a custom network.

Network Name	
Hedera	
New RPC URL	
https://testnet.hashio.io/api	
Chain ID	
296	
Currency Symbol	
HBAR	

```
<table><thead><tr><th width="140">Config</th><th
width="121.33333333333331">Default</th><th>Description</th></tr></
thead><tbody><tr><td>CHAINID</td><td><code>0x12a</code></td><td>The network
chain id. Local and previewnet envs should use <code>0x12a</code> (298).
Previewnet, Testnet and Mainnet should use <code>0x129</code> (297),
<code>0x128</code> (296) and <code>0x127</code> (295)
respectively</td></tr></tbody></table>
```

Step 3: Import your Hedera account into Metamask

Import your newly created ECDSA-based Hedera account into MetaMask using your private key from above.

```
<div>
```

```
<figure><figcaption><p>Use your private key to import your Hedera account into
Metamask</p></figcaption></figure>
```

```
<figure><figcaption><p>Your Hedera Account in Metamask</p></figcaption></figure>
```

```
</div>
```

Summary

Congratulations! 🎉 You've successfully connected your Hedera account to MetaMask! You may now send and receive HBAR on the Hedera Testnet via MetaMask!

The release of the JSON-RPC Relay to developers brings greater usability to the developer community by supporting common Ethereum developer tooling while building on Hedera.

Happy Building! Feel free to reach out if you have any questions:

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="right"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="right"><p>Writer: Abi, DevRel Engineer</p><p><a
href="https://github.com/a-ridley">GitHub</a> | <a
href="https://www.linkedin.com/in/a-ridley/">LinkedIn</a></p></td><td><a
href="https://www.linkedin.com/in/a-ridley/">https://www.linkedin.com/in/a-
ridley/</a></td></tr><tr><td align="right"><p>Editor: Krystal, Technical
Writer</p><p><a href="https://github.com/theekrystallee">GitHub</a> | <a
href="https://hashnode.com/@theekrystallee">Hashnode</a></p></td><td><a
href="https://twitter.com/theekrystallee">https://twitter.com/theekrystallee</
a></td></tr></tbody></table>
```

deploy-smart-contracts-on-hedera-using-truffle.md:

Deploy Smart Contracts on Hedera Using Truffle

The Hedera JSON RPC Relay enables developers to use their favorite EVM-compatible tools such as Truffle, Hardhat, Web3JS, EthersJS, to deploy and interact with smart contracts on the Hedera network. As highlighted in a previous article, the relay provides applications and tools seamless access to Hedera while masking implementation complexities and preventing reductions in performance, security, and scalability.

This tutorial shows you how to deploy smart contracts on Hedera using Truffle and the JSON RPC Relay with the following steps:

1. Create an account that has ECDSA keys using the Javascript SDK
2. Compile a contract using Truffle
3. Deploy the smart contract to Hedera network through the JSON RPC Relay

You can find more examples using Truffle, Web3JS, and Hardhat in this GitHub repository.

Prerequisites

Get a Hedera testnet account
This Codesandbox is already setup for you to try this example
Fork the sandbox
Remember to provide your testnet account credentials for the operator in the .env file
Open a new terminal and run:
npm install -g truffle (this installation may take a few minutes)
node create-account.js
Get the example code from GitHub

Table of Contents

1. Create an ECDSA Account
2. Compile Smart Contract
3. Deploy Smart Contract
4. Additional Resources

Create an Account that Has ECDSA Keys

Hedera supports two popular types of signature algorithms, ED25519 and ECDSA. Both are used in many blockchain platforms, including Bitcoin and Ethereum. Currently, the JSON RPC Relay only supports Hedera accounts with an alias set (i.e. public address) based on its ECDSA public key. To deploy a smart contract using Truffle, we first have to create a new account that meets these criteria. The main() function in create-account.js helps us do just that.

In case you're interested in more details about auto account creation and alias, check out the documentation and HIP-32.

```
javascript
async function main() {
  // Generate ECDSA key pair
  console.log("- Generating a new key pair... \n");
  const newPrivateKey = PrivateKey.generateECDSA();
  const newPublicKey = newPrivateKey.publicKey;
  const newAliasAccountId = newPublicKey.toAccountId(0, 0);
```



```

        console.log(- New account alias: ${newAliasAccountId} \n);
        console.log(- New private key (Hedera): ${newPrivateKey} \n);
        console.log(- New public key (Hedera): ${newPublicKey} \n);
        console.log(- New private key (RAW EVM): 0x$
{newPrivateKey.toStringRaw()} \n);
        console.log(- New public key (RAW): 0x${newPublicKey.toStringRaw()} \n);
        console.log(- New public key (EVM): 0x$
{newPublicKey.toEthereumAddress()} \n\n);

        // Transfer HBAR to newAliasAccountId to auto-create the new account
        // Get account information from a transaction record query
        const [txReceipt, txRecQuery] = await autoCreateAccountFcn(operatorId,
newAliasAccountId, 100);
        console.log(- HBAR Transfer to new account: ${txReceipt.status} \n\n);
        console.log(- Parent transaction ID: ${txRecQuery.transactionId} \n);
        console.log(- Child transaction ID: $
{txRecQuery.children[0].transactionId.toString()} \n);
        console.log(
            - New account ID (from RECORD query): $
{txRecQuery.children[0].receipt.accountId.toString()} \n
        );

        // Get account information from a mirror node query
        const mirrorQueryResult = await mirrorQueryFcn(newPublicKey);
        console.log(
            - New account ID (from MIRROR query): $
{mirrorQueryResult.data?.accounts[0].account} \n
        );
    }
}

```

Executing this code generates a new ECDSA key pair, displays the information about the keys in Hedera and EVM formats, and transfers HBAR to the account alias (newAliasAccountId) to auto-create a Hedera account that meets the criteria mentioned before. Information about the new account is obtained in two ways, a transaction record query and a mirror node query.

Console Output:

<figure><figcaption></figcaption></figure>

IMPORTANT NOTE: Private keys for Testnet are displayed here for educational purposes only. Never share your private key(s) with others, as that may result in lost funds, or loss of control over your account.

The next step is to deploy a smart contract using Truffle and the newly created Hedera account. Copy the value from “New private key (RAW EVM)” in the console output and paste it into the ETH\PRIVATE\KEY variable in the .env file (if you cloned the repository, you may need to rename the file from .env\sample to .env).

Helper Functions

The functions autoCreateAccountFcn() and mirrorQueryFcn() perform the auto account creation and mirror query, respectively.

```

javascript
async function autoCreateAccountFcn(senderAccountId, receiverAccountId,
hbarAmount) {
    //Transfer hbar to the account alias to auto-create account
    const transferToAliasTx = new TransferTransaction()

```

```

        .addHbarTransfer(senderAccountId, new Hbar(-hbarAmount))
        .addHbarTransfer(receiverAccountId, new Hbar(hbarAmount))
        .freezeWith(client);
const transferToAliasSign = await transferToAliasTx.sign(operatorKey);
const transferToAliasSubmit = await transferToAliasSign.execute(client);
const transferToAliasRx = await transferToAliasSubmit.getReceipt(client);

    // Get a transaction record and query the record to get information about
the account creation
    const transferToAliasRec = await transferToAliasSubmit.getRecord(client);
    const txRecordQuery = await new TransactionRecordQuery()
        .setTransactionId(transferToAliasRec.transactionId)
        .setIncludeChildren(true)
        .execute(client);
    return [transferToAliasRx, txRecordQuery];
}

```

```

javascript
async function mirrorQueryFcn(publicKey) {
    // Query a mirror node for information about the account creation
    await delay(10000); // Wait for 10 seconds before querying account id
    const mirrorNodeUrl = "https://testnet.mirrornode.hedera.com/api/v1/";
    const mQuery = await axios.get(
        mirrorNodeUrl + "accounts?account.publickey=" +
        publicKey.toStringRaw()
    );
    return mQuery;
}

```

Compile a Smart Contract Using Truffle

Now it's time to compile SimpleStorage, which is a basic smart contract that allows anyone to set and get data.

```

solidity
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract SimpleStorage {
    uint256 data;

    function getData() external view returns (uint256) {
        return data;
    }

    function setData(uint256 data) external {
        data = data;
    }
}

```

Use the following command to perform the compilation:

```

javascript
truffle compile

```

Console Output:

<figure>https://github.com/ed-marquez > GitHub https://www.linkedin.com/in/ed-marquez/ > LinkedIn https://www.linkedin.com/in/ed-marquez/ > https://www.linkedin.com/in/ed-marquez/ </p>
<p>Editor: Krystal, Technical Writer</p> <p> https://github.com/theekrystallee > GitHub https://hashnode.com/@theekrystallee > Hashnode https://twitter.com/theekrystallee > https://twitter.com/theekrystallee </p>

deploy-your-first-smart-contract.md:

Deploy Your First Smart Contract

In this tutorial, you will learn how to create a simple smart contract on Hedera using Solidity.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

Table of Contents

1. Create Smart Contract
2. Store Bytecode on Hedera
3. Deploy Hedera Smart Contract
4. Call Contract Functions

1. Create a "Hello Hedera" Smart Contract

Create a smart contract in solidity using the remix IDE. The "Hello Hedera" contract Solidity file is sampled below along with the "Hello Hedera" JSON file that is produced after the contract has been compiled. You can use remix to create and compile the contract yourself or you can copy the files below into your project. If you are not familiar with Solidity you can check out the docs here. Hedera supports the latest version of Solidity (v0.8.9) on previewnet and testnet.

The contract stores two variables the `owner` and `message`. The constructor passes in the `message` parameter. The

style="color:blue;">setmessage</mark> function allows the owner to update the message variable and the <mark style="color:blue;">getmessage</mark> function allows you to return the message.

The HelloHedera.sol will serve as a reference to the contract that was compiled. The HelloHedera.json file contains the <mark style="color:blue;">data.bytecode.object</mark> field that will be used to store the contract bytecode in a file on the Hedera network.

```
{% tabs %}
{% tab title="HelloHedera.sol" %}
solidity
pragma solidity >=0.7.0 <0.8.9;

contract HelloHedera {
    // the contract's owner, set in the constructor
    address owner;

    // the message we're storing
    string message;

    constructor(string memory message) {
        // set the owner of the contract for kill()
        owner = msg.sender;
        message = message;
    }

    function setmessage(string memory message) public {
        // only allow the owner to update the message
        if (msg.sender != owner) return;
        message = message;
    }

    // return a string
    function getmessage() public view returns (string memory) {
        return message;
    }

    // recover the funds of the contract
    function kill() public { if (msg.sender == owner)
selfdestruct(payable(msg.sender)); }
}

{% endtab %}

{% tab title="HelloHedera.json" %}
json
{
    "deploy": {
        "VM:-": {
            "linkReferences": {},
            "autoDeployLib": true
        },
        "main:1": {
            "linkReferences": {},
            "autoDeployLib": true
        },
        "ropsten:3": {
            "linkReferences": {},
            "autoDeployLib": true
        },
        "rinkeby:4": {
            "linkReferences": {},
            "autoDeployLib": true
        }
    }
}
```

```

    },
    "kovan:42": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "görli:5": {
      "linkReferences": {},
      "autoDeployLib": true
    },
    "Custom": {
      "linkReferences": {},
      "autoDeployLib": true
    }
  },
  "data": {
    "bytecode": {
      "linkReferences": {},
      "object":
        "608060405234801561001057600080fd5b506040516105583803806105588339818101604052602
        081101561003357600080fd5b8101908080516040519392919084640100000000821115610053576
        00080fd5b8382019150602082018581111561006957600080fd5b825186600182028301116401000
        000008211171561008657600080fd5b8083526020830192505050908051906020019080838360005
        b838110156100ba57808201518184015260208101905061009f565b50505050905090810190601f1
        680156100e75780820380516001836020036101000a031916815260200191505b506040525050503
        36000806101000a81548173fffffffffffffffffffffffffffffffffffffffff021916908373fffff
        ffffffffffffffffffffffffffffffffffffffffff160217905550806001908051906020019061014492919
        061014b565b50506101e8565b8280546001816001161561010002031660029004906000526020600
        02090601f016020900481019282601f1061018c57805160ff19168380011785556101ba565b82800
        1600101855582156101ba579182015b828111156101b957825182559160200191906001019061019
        e565b5b5090506101c791906101cb565b5090565b5b808211156101e457600081600090555060010
        16101cc565b5090565b610361806101f76000396000f3fe608060405234801561001057600080fd5
        b50600436106100365760003560e01c80632e9826021461003b57806332af2edb146100f6575b600
        080fd5b6100f46004803603602081101561005157600080fd5b81019080803590602001906401000
        0000081111561006e57600080fd5b82018360208201111561008057600080fd5b803590602001918
        460018302840111640100000000831117156100a257600080fd5b91908080601f016020809104026
        020016040519081016040528093929190818152602001838380828437600081840152601f19601f8
        20116905080830192505050505050509192919290505050610179565b005b6100fe6101ec565b604
        0518080602001828103825283818151815260200191508051906020019080838360005b838110156
        1013e578082015181840152602081019050610123565b50505050905090810190601f16801561016
        b5780820380516001836020036101000a031916815260200191505b509250505060405180910390f
        35b60008054906101000a900473fffffffffffffffffffffffffffffffffffffffff1673fffffffffff
        ffffffffffffffffffffffffffffffffff163373fffffffffffffffffffffffffffffffff161
        46101d1576101e9565b80600190805190602001906101e792919061028e565b505b50565b6060600
        18054600181600116156101000203166002900480601f01602080910402602001604051908101604
        05280929190818152602001828054600181600116156101000203166002900480156102845780601
        f1061025957610100808354040283529160200191610284565b820191906000526020600020905b8
        1548152906001019060200180831161026757829003601f168201915b5050505050905090565b828
        054600181600116156101000203166002900490600052602060002090601f0160209004810192826
        01f106102cf57805160ff19168380011785556102fd565b828001600101855582156102fd5791820
        15b828111156102fc5782518255916020019190600101906102e1565b5b50905061030a919061030
        e565b5090565b5b8082111561032757600081600090555060010161030f565b509056fea26469706
        673582212201644465f5f73dfd73a518b57770f5adb27f025842235980d7a0f4e15b1acb18e64736
        f6c63430007000033",
      "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10
      JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x40 MLOAD PUSH2 0x558 CODESIZE
      SUB DUP1 PUSH2 0x558 DUP4 CODECOPY DUP2 DUP2 ADD PUSH1 0x40 MSTORE PUSH1 0x20
      DUP2 LT ISZERO PUSH2 0x33 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1
      DUP1 DUP1 MLOAD PUSH1 0x40 MLOAD SWAP4 SWAP3 SWAP2 SWAP1 DUP5 PUSH5 0x100000000
      DUP3 GT ISZERO PUSH2 0x53 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP4 DUP3 ADD
      SWAP2 POP PUSH1 0x20 DUP3 ADD DUP6 DUP2 GT ISZERO PUSH2 0x69 JUMPI PUSH1 0x0
      DUP1 REVERT JUMPDEST DUP3 MLOAD DUP7 PUSH1 0x1 DUP3 MUL DUP4 ADD GT PUSH5
      0x1000000000 DUP3 GT OR ISZERO PUSH2 0x86 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST
      DUP1 DUP4 MSTORE PUSH1 0x20 DUP4 ADD SWAP3 POP POP POP SWAP1 DUP1 MLOAD SWAP1
      PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2

```

0xBA JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1
POP PUSH2 0x9F JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1
PUSH1 0x1F AND DUP1 ISZERO PUSH2 0xE7 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1
DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2
POP JUMPDEST POP PUSH1 0x40 MSTORE POP POP POP CALLER PUSH1 0x0 DUP1 PUSH2 0x100
EXP DUP2 SLOAD DUP2 PUSH20 0xFF MUL NOT
AND SWAP1 DUP4 PUSH20 0xFF AND MUL OR
SWAP1 SSTORE POP DUP1 PUSH1 0x1 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1
PUSH2 0x144 SWAP3 SWAP2 SWAP1 PUSH2 0x14B JUMP JUMPDEST POP POP PUSH2 0x1E8 JUMP
JUMPDEST DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100 MUL SUB
AND PUSH1 0x2 SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256
SWAP1 PUSH1 0x1F ADD PUSH1 0x20 SWAP1 DIV DUP2 ADD SWAP3 DUP3 PUSH1 0x1F LT
PUSH2 0x18C JUMPI DUP1 MLOAD PUSH1 0xFF NOT AND DUP4 DUP1 ADD OR DUP6 SSTORE
PUSH2 0x1BA JUMP JUMPDEST DUP3 DUP1 ADD PUSH1 0x1 ADD DUP6 SSTORE DUP3 ISZERO
PUSH2 0x1BA JUMPI SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2 GT ISZERO PUSH2 0x1B9 JUMPI
DUP3 MLOAD DUP3 SSTORE SWAP2 PUSH1 0x20 ADD SWAP2 SWAP1 PUSH1 0x1 ADD SWAP1
PUSH2 0x19E JUMP JUMPDEST JUMPDEST POP SWAP1 POP PUSH2 0x1C7 SWAP2 SWAP1 PUSH2
0x1CB JUMP JUMPDEST POP SWAP1 JUMP JUMPDEST JUMPDEST DUP1 DUP3 GT ISZERO PUSH2
0x1E4 JUMPI PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP PUSH1 0x1 ADD PUSH2 0x1CC
JUMP JUMPDEST POP SWAP1 JUMP JUMPDEST PUSH2 0x361 DUP1 PUSH2 0x1F7 PUSH1 0x0
CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1
ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4
CALLDATASIZE LT PUSH2 0x36 JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1
PUSH4 0x2E982602 EQ PUSH2 0x3B JUMPI DUP1 PUSH4 0x32AF2EDB EQ PUSH2 0xF6 JUMPI
JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH2 0xF4 PUSH1 0x4 DUP1 CALLDATASIZE
SUB PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x51 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST
DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH5
0x1000000000 DUP2 GT ISZERO PUSH2 0x6E JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP3
ADD DUP4 PUSH1 0x20 DUP3 ADD GT ISZERO PUSH2 0x80 JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP2 DUP5 PUSH1 0x1 DUP4 MUL
DUP5 ADD GT PUSH5 0x1000000000 DUP4 GT OR ISZERO PUSH2 0xA2 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST SWAP2 SWAP1 DUP1 DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV
MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP4
SWAP3 SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP4 DUP4 DUP1 DUP3 DUP5
CALLDATACOPY PUSH1 0x0 DUP2 DUP5 ADD MSTORE PUSH1 0x1F NOT PUSH1 0x1F DUP3 ADD
AND SWAP1 POP DUP1 DUP4 ADD SWAP3 POP POP POP POP POP POP POP SWAP2 SWAP3 SWAP2
SWAP3 SWAP1 POP POP POP PUSH2 0x179 JUMP JUMPDEST STOP JUMPDEST PUSH2 0xFE PUSH2
0x1EC JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3
MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD
SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO
PUSH2 0x13E JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD
SWAP1 POP PUSH2 0x123 JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD
SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH2 0x16B JUMPI DUP1 DUP3 SUB DUP1 MLOAD
PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20
ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB
SWAP1 RETURN JUMPDEST PUSH1 0x0 DUP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV
PUSH20 0xFF AND PUSH20
0xFF AND CALLER PUSH20
0xFF AND EQ PUSH2 0x1D1 JUMPI PUSH2 0x1E9
JUMP JUMPDEST DUP1 PUSH1 0x1 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH2
0x1E7 SWAP3 SWAP2 SWAP1 PUSH2 0x28E JUMP JUMPDEST POP JUMPDEST POP JUMP JUMPDEST
PUSH1 0x60 PUSH1 0x1 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100
MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV
MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3
SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1
0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2
0x284 JUMPI DUP1 PUSH1 0x1F LT PUSH2 0x259 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV
MUL DUP4 MSTORE SWAP2 PUSH1 0x20 ADD SWAP2 PUSH2 0x284 JUMP JUMPDEST DUP3 ADD
SWAP2 SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2
SLOAD DUP2 MSTORE SWAP1 PUSH1 0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2
0x267 JUMPI DUP3 SWAP1 SUB PUSH1 0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP
POP POP SWAP1 POP SWAP1 JUMP JUMPDEST DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1
AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE
PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 PUSH1 0x1F ADD PUSH1 0x20 SWAP1 DIV DUP2

```

ADD SWAP3 DUP3 PUSH1 0x1F LT PUSH2 0x2CF JUMPI DUP1 MLOAD PUSH1 0xFF NOT AND
DUP4 DUP1 ADD OR DUP6 SSTORE PUSH2 0x2FD JUMP JUMPDEST DUP3 DUP1 ADD PUSH1 0x1
ADD DUP6 SSTORE DUP3 ISZERO PUSH2 0x2FD JUMPI SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2
GT ISZERO PUSH2 0x2FC JUMPI DUP3 MLOAD DUP3 SSTORE SWAP2 PUSH1 0x20 ADD SWAP2
SWAP1 PUSH1 0x1 ADD SWAP1 PUSH2 0x2E1 JUMP JUMPDEST JUMPDEST POP SWAP1 POP PUSH2
0x30A SWAP2 SWAP1 PUSH2 0x30E JUMP JUMPDEST POP SWAP1 JUMP JUMPDEST JUMPDEST
DUP1 DUP3 GT ISZERO PUSH2 0x327 JUMPI PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP
PUSH1 0x1 ADD PUSH2 0x30F JUMP JUMPDEST POP SWAP1 JUMP INVALID LOG2 PUSH5
0x6970667358 0x22 SLT KECCAK256 AND DIFFICULTY CHAINID 0x5F 0x5F PUSH20
0xDfD73A518B57770F5ADB27F025842235980D7A0F 0x4E ISZERO 0xB1 0xAC 0xB1 DUP15
PUSH5 0x736F6C6343 STOP SMOD STOP STOP CALLER ",
"sourceMap":
"33:623:0::-:0;;;186:160;;;;;
;
;
;
331:8;321:7;;18;:::i;::-;186:160;33:623;
;
;
::0;::-;
},
"deployedBytecode": {
  "immutableReferences": {},
  "linkReferences": {},
  "object":
"608060405234801561001057600080fd5b50600436106100365760003560e01c80632e982602146
1003b57806332af2edb146100f6575b600080fd5b6100f4600480360360208110156100515760008
0fd5b810190808035906020019064010000000081111561006e57600080fd5b82018360208201111
561008057600080fd5b803590602001918460018302840111640100000000831117156100a257600
080fd5b91908080601f0160208091040260200160405190810160405280939291908181526020018
38380828437600081840152601f19601f82011690508083019250505050505050919291929050505
0610179565b005b6100fe6101ec565b6040518080602001828103825283818151815260200191508
051906020019080838360005b8381101561013e578082015181840152602081019050610123565b5
0505050905090810190601f16801561016b5780820380516001836020036101000a0319168152602
00191505b509250505060405180910390f35b60008054906101000a900473fffffffffffffffffff
ffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373fffffffffff
ffffffffffffffffffffffffffffffff16146101d1576101e9565b80600190805190602001906101e
792919061028e565b505b50565b606060018054600181600116156101000203166002900480601f0
16020809104026020016040519081016040528092919081815260200182805460018160011615610
1000203166002900480156102845780601f106102595761010080835404028352916020019161028
4565b820191906000526020600020905b81548152906001019060200180831161026757829003601
f168201915b5050505050905090565b8280546001816001161561010002031660029004906000526
02060002090601f016020900481019282601f106102cf57805160ff19168380011785556102fd565
b828001600101855582156102fd579182015b828111156102fc5782518259160200191906001019
06102e1565b5b50905061030a919061030e565b5090565b5b8082111561032757600081600090555
060010161030f565b509056fea26469706673582212201644465f5f73dfd73a518b57770f5adb27f
025842235980d7a0f4e15b1acb18e64736f6c63430007000033",
"opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10
JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE LT PUSH2 0x36
JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x2E982602 EQ PUSH2 0x3B
JUMPI DUP1 PUSH4 0x32AF2EDB EQ PUSH2 0xF6 JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT
JUMPDEST PUSH2 0xF4 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO
PUSH2 0x51 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1
CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH5 0x100000000 DUP2 GT ISZERO PUSH2
0x6E JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP3 ADD DUP4 PUSH1 0x20 DUP3 ADD GT
ISZERO PUSH2 0x80 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP1 CALLDATALOAD SWAP1
PUSH1 0x20 ADD SWAP2 DUP5 PUSH1 0x1 DUP4 MUL DUP5 ADD GT PUSH5 0x100000000 DUP4
GT OR ISZERO PUSH2 0xA2 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST SWAP2 SWAP1 DUP1
DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV MUL PUSH1 0x20 ADD PUSH1 0x40
MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP4 SWAP3 SWAP2 SWAP1 DUP2 DUP2
MSTORE PUSH1 0x20 ADD DUP4 DUP4 DUP1 DUP3 DUP5 CALLDATACOPY PUSH1 0x0 DUP2 DUP5
ADD MSTORE PUSH1 0x1F NOT PUSH1 0x1F DUP3 ADD AND SWAP1 POP DUP1 DUP4 ADD SWAP3
POP POP POP POP POP POP SWAP2 SWAP3 SWAP2 SWAP3 SWAP1 POP POP POP PUSH2
0x179 JUMP JUMPDEST STOP JUMPDEST PUSH2 0xFE PUSH2 0x1EC JUMP JUMPDEST PUSH1
0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2
MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1

```



```

DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x13E JUMPI DUP1
DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0x123
JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1
ISZERO PUSH2 0x16B JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB
PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP
SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1
0x0 DUP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND CALLER PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ PUSH2 0x1D1 JUMPI PUSH2 0x1E9
JUMP JUMPDEST DUP1 PUSH1 0x1 SWAP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 PUSH2
0x1E7 SWAP3 SWAP2 SWAP1 PUSH2 0x28E JUMP JUMPDEST POP JUMPDEST POP JUMP JUMPDEST
PUSH1 0x60 PUSH1 0x1 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1 AND ISZERO PUSH2 0x100
MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 PUSH1 0x1F ADD PUSH1 0x20 DUP1 SWAP2 DIV
MUL PUSH1 0x20 ADD PUSH1 0x40 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 SWAP3
SWAP2 SWAP1 DUP2 DUP2 MSTORE PUSH1 0x20 ADD DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1
0x1 AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV DUP1 ISZERO PUSH2
0x284 JUMPI DUP1 PUSH1 0x1F LT PUSH2 0x259 JUMPI PUSH2 0x100 DUP1 DUP4 SLOAD DIV
MUL DUP4 MSTORE SWAP2 PUSH1 0x20 ADD SWAP2 PUSH2 0x284 JUMP JUMPDEST DUP3 ADD
SWAP2 SWAP1 PUSH1 0x0 MSTORE PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 JUMPDEST DUP2
SLOAD DUP2 MSTORE SWAP1 PUSH1 0x1 ADD SWAP1 PUSH1 0x20 ADD DUP1 DUP4 GT PUSH2
0x267 JUMPI DUP3 SWAP1 SUB PUSH1 0x1F AND DUP3 ADD SWAP2 JUMPDEST POP POP POP
POP POP SWAP1 POP SWAP1 JUMP JUMPDEST DUP3 DUP1 SLOAD PUSH1 0x1 DUP2 PUSH1 0x1
AND ISZERO PUSH2 0x100 MUL SUB AND PUSH1 0x2 SWAP1 DIV SWAP1 PUSH1 0x0 MSTORE
PUSH1 0x20 PUSH1 0x0 KECCAK256 SWAP1 PUSH1 0x1F ADD PUSH1 0x20 SWAP1 DIV DUP2
ADD SWAP3 DUP3 PUSH1 0x1F LT PUSH2 0x2CF JUMPI DUP1 MLOAD PUSH1 0xFF NOT AND
DUP4 DUP1 ADD OR DUP6 SSTORE PUSH2 0x2FD JUMP JUMPDEST DUP3 DUP1 ADD PUSH1 0x1
ADD DUP6 SSTORE DUP3 ISZERO PUSH2 0x2FD JUMPI SWAP2 DUP3 ADD JUMPDEST DUP3 DUP2
GT ISZERO PUSH2 0x2FC JUMPI DUP3 MLOAD DUP3 SSTORE SWAP2 PUSH1 0x20 ADD SWAP2
SWAP1 PUSH1 0x1 ADD SWAP1 PUSH2 0x2E1 JUMP JUMPDEST JUMPDEST POP SWAP1 POP PUSH2
0x30A SWAP2 SWAP1 PUSH2 0x30E JUMP JUMPDEST POP SWAP1 JUMP JUMPDEST JUMPDEST
DUP1 DUP3 GT ISZERO PUSH2 0x327 JUMPI PUSH1 0x0 DUP2 PUSH1 0x0 SWAP1 SSTORE POP
PUSH1 0x1 ADD PUSH2 0x30F JUMP JUMPDEST POP SWAP1 JUMP INVALID LOG2 PUSH5
0x6970667358 0x22 SLT KECCAK256 AND DIFFICULTY CHAINID 0x5F 0x5F PUSH20
0xDFD73A518B57770F5ADB27F025842235980D7A0F 0x4E ISZERO 0xB1 0xAC 0xB1 DUP15
PUSH5 0x736F6C6343 STOP SMOD STOP STOP CALLER ",
"sourceMap":
"33:623:0::-:0;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;352:182;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;:i;::-;563:90;:::i;::-;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;352:182;486:5;::::;472:19;:10;:19;;468:32;493:7;468:32;519:8
;509:7;:18;:::::i;::-;352:182;:::o;563:90::-;607:13;639:7;632:14;::;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;563:90;:::o;-1:-1:-
1:-;
1:-;
:::i;::-;:::o;::-;:::o"
},
"gasEstimates": {
  "creation": {
    "codeDepositCost": "173000",
    "executionCost": "infinite",
    "totalCost": "infinite"
  },
  "external": {
    "getMessage()": "infinite",
    "setMessage(string)": "infinite"
  }
},
"methodIdentifiers": {
  "getMessage()": "32af2edb",
  "setMessage(string)": "2e982602"
}
},

```

```

"abi": [
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "message",
        "type": "string"
      }
    ],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "inputs": [],
    "name": "getMessage",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "string",
        "name": "message",
        "type": "string"
      }
    ],
    "name": "setMessage",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
}

{% endtab %}
{% endtabs %}

```

2. Store the Smart Contract Bytecode on Hedera

Create a file using the `FileCreateTransaction()` API to store the hex-encoded byte code of the "Hello Hedera" contract. Once the file is created, you can obtain the file ID from the receipt of the transaction.

You can alternatively use the `CreateContractFlow()` API that creates the bytecode file for you and subsequently creates the contract on Hedera in a single API.

{% hint style="warning" %}

Note: The bytecode is required to be hex-encoded. It should not be the actual data the hex represents.

{% endhint %}

{% tabs %}

```

{% tab title="Java" %}
java
//Import the compiled contract from the HelloHedera.json file
Gson gson = new Gson();
JsonObject jsonObject;

InputStream jsonStream =
HelloHederaSmartContract.class.getClassLoader().getResourceAsStream("HelloHedera
.json");
jsonObject = gson.fromJson(new InputStreamReader(jsonStream,
StandardCharsets.UTF8), JsonObject.class);

//Store the "object" field from the HelloHedera.json file as hex-encoded
bytecode
String object =
jsonObject.getAsJsonObject("data").getAsJsonObject("bytecode").get("object").get
AsString();
byte[] bytecode = object.getBytes(StandardCharsets.UTF8);

//Create a file on Hedera and store the hex-encoded bytecode
FileCreateTransaction fileCreateTx = new FileCreateTransaction()
    //Set the bytecode of the contract
    .setContents(bytecode);

//Submit the file to the Hedera test network signing with the transaction fee
payer key specified with the client
TransactionResponse submitTx = fileCreateTx.execute(client);

//Get the receipt of the file create transaction
TransactionReceipt fileReceipt = submitTx.getReceipt(client);

//Get the file ID from the receipt
FileId bytecodeFileId = fileReceipt.fileId;

//Log the file ID
System.out.println("The smart contract bytecode file ID is " +bytecodeFileId)

//v2 Hedera Java SDK

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Import the compiled contract from the HelloHedera.json file
let helloHedera = require("./HelloHedera.json");
const bytecode = helloHedera.data.bytecode.object;

//Create a file on Hedera and store the hex-encoded bytecode
const fileCreateTx = new FileCreateTransaction()
    //Set the bytecode of the contract
    .setContents(bytecode);

//Submit the file to the Hedera test network signing with the transaction fee
payer key specified with the client
const submitTx = await fileCreateTx.execute(client);

//Get the receipt of the file create transaction
const fileReceipt = await submitTx.getReceipt(client);

//Get the file ID from the receipt
const bytecodeFileId = fileReceipt.fileId;

//Log the file ID
console.log("The smart contract byte code file ID is " +bytecodeFileId)

```

```

{% endtab %}

{% tab title="Go" %}
go
//Import and parse the compiled contract from the HelloHedera.json file
helloHedera, err := ioutil.ReadFile("./HelloHedera.json")
if err != nil {
    println(err.Error(), ": error reading HelloHedera.json")
    return
}

var contract contract = contract{}

err = json.Unmarshal([]byte(helloHedera), &contract)
if err != nil {
    println(err.Error(), ": error unmarshaling the json file")
    return
}

bytecode := []byte(contract.Object)

//Create a file on Hedera and store the hex-encoded bytecode
fileTx, err := hedera.NewFileCreateTransaction().
    //Set the bytecode of the contract
    SetContents([]byte(bytecode)).
    //Submit the transaction to a Hedera network
    Execute(client)

if err != nil {
    println(err.Error(), ": error creating file")
    return
}

//Get the receipt of the file create transaction
fileReceipt, err := fileTx.GetReceipt(client)
if err != nil {
    println(err.Error(), ": error getting file create transaction receipt")
    return
}

//Get the file ID
byteCodefileID := fileReceipt.FileID

//Log the file ID
fmt.Printf("contract bytecode file: %v\n", byteCodefileID)

{% endtab %}
{% endtabs %}

```

3. Deploy a Hedera Smart Contract

Create the contract and set the file ID to the file ID that stores the hex-encoded byte code from the previous step. You will also need to set gas the value that will create the contract and pass the constructor parameters using `<mark style="color:blue;">ContractFunctionParameters()</mark> API<mark style="color:purple;">.</mark>` In this example, "hello from Hedera!" was passed to the constructor. After the transaction is successfully executed, you can get the contract ID from the receipt.

```
{% hint style="warning" %}
```

Note: You will need to set the gas value high enough to deploy the contract. If

```

you don't have enough gas, you will receive an INSUFFICIENTGAS response.
{% endhint %}

{% tabs %}
{% tab title="Java" %}
java
// Instantiate the contract instance
ContractCreateTransaction contractTx = new ContractCreateTransaction()
    //Set the file ID of the Hedera file storing the bytecode
    .setBytecodeFileId(newFileId)
    //Set the gas to instantiate the contract
    .setGas(1000000)
    //Provide the constructor parameters for the contract
    .setConstructorParameters(new
ContractFunctionParameters().addString("Hello from Hedera!"));

//Submit the transaction to the Hedera test network
TransactionResponse contractResponse = contractTx.execute(client);

//Get the receipt of the file create transaction
TransactionReceipt contractReceipt = contractResponse.getReceipt(client);

//Get the smart contract ID
ContractId newContractId = contractReceipt.contractId;

//Log the smart contract ID
System.out.println("The smart contract ID is " + newContractId);

//v2 Hedera Java SDK

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Instantiate the contract instance
const contractTx = await new ContractCreateTransaction()
    //Set the file ID of the Hedera file storing the bytecode
    .setBytecodeFileId(bytecodeFileId)
    //Set the gas to instantiate the contract
    .setGas(1000000)
    //Provide the constructor parameters for the contract
    .setConstructorParameters(new
ContractFunctionParameters().addString("Hello from Hedera!"));

//Submit the transaction to the Hedera test network
const contractResponse = await contractTx.execute(client);

//Get the receipt of the file create transaction
const contractReceipt = await contractResponse.getReceipt(client);

//Get the smart contract ID
const newContractId = contractReceipt.contractId;

//Log the smart contract ID
console.log("The smart contract ID is " + newContractId);

//v2 JavaScript SDK

{% endtab %}

{% tab title="Go" %}
go
// Instantiate the contract instance
contractTx, err := hedera.NewContractCreateTransaction().

```

```

        //Set the file ID of the Hedera file storing the bytecode
        SetBytecodeFileID(bytecodeFileID).
        //Set the gas to instantiate the contract
        SetGas(1000000).
        //Provide the constructor parameters for the contract
        SetConstructorParameters(hedera.NewContractFunctionParameters()).
            AddString("Hello from hedera")).
        Execute(client)

if err != nil {
    println(err.Error(), ": error creating contract")
    return
}

//Get the receipt of the contract create transaction
contractReceipt, err := contractTx.GetReceipt(client)
if err != nil {
    println(err.Error(), ": error retrieving contract creation receipt")
    return
}

//Get the contract ID from the receipt
newContractID := contractReceipt.ContractID

//v2 Hedera Go SDK

{% endtab %}
{% endtabs %}

```

4. Call Contract Functions

Call the `getMessage` contract function

In the previous step, the contract message variable was set to "hello from Hedera!". You can return this message from the contract by submitting a query that will return the stored message string. The `ContractCallQuery` similarly does not modify the state of the contract like other Hedera queries. It only reads stored values.

```

{% tabs %}
{% tab title="Java" %}
java
// Calls a function of the smart contract
ContractCallQuery contractQuery = new ContractCallQuery()
    //Set the gas for the query
    .setGas(1000000)
    //Set the contract ID to return the request for
    .setContractId(newContractId)
    //Set the function of the contract to call
    .setFunction("getMessage" )
    //Set the query payment for the node returning the request
    //This value must cover the cost of the request otherwise will fail
    .setQueryPayment(new Hbar(2));

//Submit to a Hedera network
ContractFunctionResult getMessage = contractQuery.execute(client);
//Get the message
String message = getMessage.getString(0);

//Log the message
System.out.println("The contract message: " + message);

```

```

//v2 Hedera Java SDK

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Calls a function of the smart contract
const contractQuery = await new ContractCallQuery()
    //Set the gas for the query
    .setGas(1000000)
    //Set the contract ID to return the request for
    .setContractId(newContractId)
    //Set the contract function to call
    .setFunction("getMessage" )
    //Set the query payment for the node returning the request
    //This value must cover the cost of the request otherwise will fail
    .setQueryPayment(new Hbar(2));

//Submit to a Hedera network
const getMessage = await contractQuery.execute(client);

// Get a string from the result at index 0
const message = getMessage.getString(0);

//Log the message
console.log("The contract message: " + message);

//v2 Hedera JavaScript SDK

{% endtab %}

{% tab title="Go" %}
go
// Calls a function of the smart contract
contractQuery, err := hedera.NewContractCallQuery().
    //Set the contract ID to return the request for
    SetContractID(newContractID).
    //Set the gas for the query
    SetGas(1000000).
    //Set the query payment for the node returning the request
    //This value must cover the cost of the request otherwise will fail
    SetQueryPayment(hedera.NewHbar(2)).
    //Set the contract function to call
    SetFunction("getMessage", nil). // nil -> no parameters
    //Submit the query to a Hedera network
    Execute(client)

if err != nil {
    println(err.Error(), ": error executing contract call query")
    return
}

// Get a string from the result at index 0
getMessage := contractQuery.GetString(0)

//Log the message
fmt.Printf("The contract message: ", getMessage)

//v2 Hedera Go SDK

{% endtab %}
{% endtabs %}

Call the <mark style="color:purple;">setmessage</mark> contract function

```

Call the `setMessage` function of the contract. To do this you will need to use the `ContractExecuteTransaction` API. This transaction will update the contract message. Once the transaction is successfully submitted you can verify the message was updated by requesting `ContractCallQuery`. The message returned from the contract should now log "Hello from Hedera again!"

```
{% tabs %}
{% tab title="Java" %}
java
//Create the transaction to update the contract message
ContractExecuteTransaction contractExecTx = new ContractExecuteTransaction()
    //Set the ID of the contract
    .setContractId(newContractId)
    //Set the gas for the call
    .setGas(100000)
    //Set the function of the contract to call
    .setFunction("setMessage", new
ContractFunctionParameters().addString("Hello from Hedera again!"));

//Submit the transaction to a Hedera network and store the response
TransactionResponse submitExecTx = contractExecTx.execute(client);

//Get the receipt of the transaction
TransactionReceipt receipt2 = submitExecTx.getReceipt(client);

//Confirm the transaction was executed successfully
System.out.println("The transaction status is" +receipt2.status);

//Query the contract for the contract message
ContractCallQuery contractCallQuery = new ContractCallQuery()
    //Set ID of the contract to query
    .setContractId(newContractId)
    //Set the gas to execute the contract call
    .setGas(100000)
    //Set the contract function
    .setFunction("getMessage")
    //Set the query payment for the node returning the request
    //This value must cover the cost of the request otherwise will fail
    .setQueryPayment(new Hbar(2));

//Submit the query to a Hedera network
ContractFunctionResult contractUpdateResult = contractCallQuery.execute(client);

//Get the updated message
String message2 = contractUpdateResult.getString(0);

//Log the updated message
System.out.println("The contract updated message: " + message2);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the transaction to update the contract message
const contractExecTx = await new ContractExecuteTransaction()
    //Set the ID of the contract
    .setContractId(newContractId)
    //Set the gas for the contract call
    .setGas(100000)
    //Set the contract function to call
    .setFunction("setMessage", new
```



```

ContractFunctionParameters().addString("Hello from Hedera again!"));

//Submit the transaction to a Hedera network and store the response
const submitExecTx = await contractExecTx.execute(client);

//Get the receipt of the transaction
const receipt2 = await submitExecTx.getReceipt(client);

//Confirm the transaction was executed successfully
console.log("The transaction status is " +receipt2.status.toString());

//Query the contract for the contract message
const contractCallQuery = new ContractCallQuery()
    //Set the ID of the contract to query
    .setContractId(newContractId)
    //Set the gas to execute the contract call
    .setGas(100000)
    //Set the contract function to call
    .setFunction("getMessage")
    //Set the query payment for the node returning the request
    //This value must cover the cost of the request otherwise will fail
    .setQueryPayment(new Hbar(2));

//Submit the transaction to a Hedera network
const contractUpdateResult = await contractCallQuery.execute(client);

//Get the updated message at index 0
const message2 = contractUpdateResult.getString(0);

//Log the updated message to the console
console.log("The updated contract message: " + message2);

//v2 Hedera JavaScript SDK

{% endtab %}

{% tab title="Go" %}
go
//Create the transaction to update the contract message
contractExecTx, err := hedera.NewContractExecuteTransaction().
    //Set the ID of the contract
    SetContractID(newContractID).
    //Set the gas to execute the call
    SetGas(100000).
    //Set the contract function to call
    SetFunction("setMessage", hedera.NewContractFunctionParameters().
        AddString("Hello from Hedera again!")).
    Execute(client)

if err != nil {
    println(err.Error(), ": error executing contract")
    return
}

//Get the receipt of the transaction
receipt2, err := contractExecTx.GetReceipt(client)

//Confirm the transaction was executed successfully
fmt.Printf("The transaction status is", receipt2.Status))

//Query the contract for the contract message
contractCallQuery, err := hedera.NewContractCallQuery().
    //Set the contract ID to query
    SetContractID(newContractID).

```

```

//Set the gas to execute the contract call
SetGas(100000).
//Set the query payment for the node returning the request
//This value must cover the cost of the request otherwise will fail
SetQueryPayment(hedera.NewHbar(2)).
//Set the contract function to call
SetFunction("getMessage", nil). // nil -> no parameters
//Submit the query to a Hedera network node
Execute(client)

if err != nil {
    println(err.Error(), ": error executing contract call query")
    return
}

```

```

// Get a string from the result at index 0
getMessage2 := contractCallQuery.GetString(0)

```

```

//Log the message
fmt.Printf("The updated contract message: ", getMessage2)

```

```

//v2 Hedera Go SDK

```

```

{% endtab %}
{% endtabs %}

```

Congratulations :tada:! You have completed the following:

```

Created a simple smart contract on Hedera
Interacted with contract functions

```

```

{% embed url="https://www.youtube.com/watch?amp%3Bt=24s&v=L9Tm6ynayY" %}
Video tutorial
{% endembed %}

```

Additional Resources

 Have a question? Ask on StackOverflow

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden></th><th data-hidden></th><th data-hidden
data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Simi, Sr. Software Manager </p><p><a
href="https://github.com/ed-marquez">GitHub</a> | <a
href="https://www.linkedin.com/in/shunjan">LinkedIn</a></p></td><td></td><td></td></td></td><a href="https://www.linkedin.com/in/shunjan">https://www.linkedin.com/
in/shunjan</a></td></tr></tbody></table>

```

how-to-verify-a-smart-contract-on-hashscan.md:

```

---
description: >-
    How to verify a smart contract using the HashScan Smart Contract Verification
    tool.
---

```

How to Verify a Smart Contract on HashScan

Verifying smart contracts helps ensure the deployed bytecode matches the expected source files. HashScan Smart Contract Verifier is a tool that simplifies this process. This guide will walk through the basic steps of smart

contract verification using the HashScan Smart Contract Verifier tool.

<figure><figcaption><p>Smart Contract Verification Flow</p></figcaption></figure>

{% hint style="info" %}

🔔 Note: This is an initial release. API functionalities will see enhancements in upcoming updates of the <https://github.com/hashgraph/hedera-sourcify> repository.

{% endhint %}

Prerequisites

Solidity source code file of the deployed smart contract.

Solidity JSON (metadata) file of the deployed smart contract.

EVM address of the smart contract deployed on the Hedera network.

Table of Contents

1. Find the Contract
2. Import Source Files
3. Verify Contract
4. Verification Match
5. View Verified Contract
6. Re-Verify Smart Contract
7. Additional Resources

Step 1: Find the Contract on HashScan

Open a web browser and navigate to HashScan. Make sure you are on the correct Hedera network (Mainnet, Testnet, or Previewnet), and search for the deployed contract address in the search bar at the top of the page. In the Contract Bytecode section click on Verify Contract. The source code file importer popup window will open.

<figure><figcaption></figcaption></figure>

Step 2: Import Source Files

Add your Solidity source code files in the source file importer popup. Source files include the smart contract (.sol) source code file and metadata (.json) file. The metadata file can be found in the artifacts/ directory of your smart contract project and its name correlates with the smart contract. For example, the metadata for the HelloHedera.sol contract would be called HelloHedera.json.

<figure><figcaption><p>HashScan Verification Source File Importer Popup</p></figcaption></figure>

<details>

<summary>🔔 Different compiling tools require specific verification source files. Here's a brief outline of what is needed for popular tools: </summary>

1. Remix:

Required for Full Match Verification: Both the metadata file found in the contracts/artifacts/ folder and the smart contract's Solidity file. More details [here](#);

2. Hardhat:

Required for Full Match Verification: Only the output of the compilation JSON file found in the /artifacts/build-info/ folder. More details [here](#).

3. Solidity Compiler (solc):

Required for Full Match Verification: Both the metadata file (generated by solc --metadata) and the smart contract's Solidity file. More details [here](#);

4. Foundry:

Required for Full Match Verification: Both the metadata file (generated by forge-build) and the smart contract's Solidity file.

Note: Uploading only the Solidity file without the metadata file will result in a Partial Match.

</details>

Step 3: Verify Contract

After importing the source files, if you get the "Contract \<contract name> is ready to be verified" message, click VERIFY to initiate the verification process. Sourcify will then compare the deployed contract bytecode to the source files you imported in the previous step.

<figure><figcaption></figcaption></figure>

Step 4: Verification Match

If your verification is successful, the verifier will return either a <mark style="color:green;">Full Match</mark> or <mark style="color:green;">Partial Match</mark> status. Let's review each verification status and what they mean:

Full Match: Indicates the bytecode is a full (perfect) match, including all the metadata. The contract source code and metadata settings are identical to the deployed version.

Partial Match: Indicates the bytecode mostly (partially) matches with the deployed contract, except for the metadata hash like comments or variable names. It is usually sufficient for most verification purposes.

<figure><figcaption></figcaption></figure>

To learn more about each verification match status, head over to the official Sourcify documentation [here](#);

Step 5: View Verified Contract

To view the verified contract repository, click View Contract Sources in the Contract Bytecode section on HashScan. This will open a verified contract repository search page window.

<figure><figcaption></figcaption></figure>

A summary of your contract's verification details will be displayed in the new

window. Verification details include the contract address, source code files, match type, chain ID, metadata, and an option to open the repository in Remix.

<figure><figcaption><p>Verified contract repository</p></figcaption></figure>

To be directed to the Sourcify Contract Repository search page, click on  296.

<figure><figcaption><p>Sourcify Contract Repository search page</p></figcaption></figure>

Congratulations! 🎉 You have successfully learned how to verify a smart contract. Feel free to reach out on Discord if you have any questions!

Step 6: Re-Verify Smart Contract

If you change your contract or want to upgrade your contract from a Partial Match to a Full Match, there are two options for re-verification:

Option 1

Head to the smart contract verifier page and import your new updated source files.

<figure><figcaption></figcaption></figure>

Enter the smart contract address and chain, then click Verify.






<figure><figcaption></figcaption></figure>

Option 2

Revisit Step 1 and use the Re-verify Contract flow. Then proceed to Steps 2 through 5.

<figure><figcaption></figcaption></figure>

Additional Resources

-  [HashScan Network Explorer](#)
-  [Smart Contract Verifier Page](#)
-  [Verified Contract Repository](#)
-  [Sourcify Documentation](#)
-  [Smart Contract Documentation](#)

<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Krystal, Technical Writer</p><p>GitHub | Twitter</p></td><td><a

https://twitter.com/theekrystallee	https://twitter.com/theekrystallee
<p>Editor: Nana, Sr. Software Manager</p> <p> GitHub LinkedIn </p>	<p>Editor: Ed, DevRel Engineer</p> <p> GitHub LinkedIn </p>
<p>Editor: Logan, Software Engineer Intern</p> <p> GitHub LinkedIn </p>	<p> https://github.com/quiet-node </p>

README.md:

Smart Contracts Service

send-and-receive-hbar-using-solidity-smart-contracts.md:

Send and Receive HBAR Using Solidity Smart Contracts

Smart contracts on Hedera can hold and exchange value in the form of HBAR, Hedera Token Service (HTS) tokens, and even ERC tokens. This is fundamental for building decentralized applications that rely on contracts in areas like DeFi, ESG, NFT marketplaces, DAOs, and more.

Let's learn how to send and receive HBAR to and from Hedera contracts. Part 1 of the series focused on using the Hedera SDKs. This second part goes over transferring HBAR to and from contracts using Solidity.

Follow these main 3 steps:

1. Create the Hedera accounts needed for testing and deploy a smart contract on the Testnet
2. Move HBAR to the contract using fallback and receive functions, a payable function, and the SDK
3. Move HBAR from the contract to Alice using the transfer, send, and call methods

Throughout the tutorial, you also learn how to check the HBAR balance of the contract by calling a function of the contract itself and by using the SDK query. The last step is to review the transaction history for the contract and the operator account in a mirror node explorer, like HashScan.



Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

Table of Contents

1. Create Accounts and Deploy a Contract

2. Get HBAR to  Contract
3. Get HBAR from  Contract
4. Summary
5. Additional Resources

Create Accounts and Deploy a Contract

This example involves 3 Hedera accounts, 1 contract, and 1 Hedera Token Service (HTS) token. The Operator account (your Testnet account credentials) is used to build the Hedera client to submit transactions to the Hedera network – that's the first account. The Treasury and Alice are new accounts (created by the Operator) to represent additional parties in your test – those are the second and third accounts respectively.

A portion of the application file (index.js) and the entire Solidity contract (hbarToAndFromContract.sol) are shown in the tabs below.

The Solidity file has functions for getting HBAR to the contract (receive, fallback, tokenAssociate), getting HBAR from the contract (transferHbar, sendHbar, callHbar), and checking the HBAR balance of the contract (getBalance).

This portion of index.js configures and creates the accounts, deploys the contract, and stores the HTS token ID. The functions accountCreatorFcn and contractDeployFcn create new accounts and deploy the contract to the network, respectively. These functions simplify the account creation and contract deployment process and are reusable in case you need them in the future. This modular approach is used throughout the tutorial.

```
{% tabs %}
{% tab title="Index.Js" %}
javascript
// Configure accounts and client
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromString(process.env.OPERATORPVKEY);
const client = Client.forTestnet().setOperator(operatorId, operatorKey);

async function main() {
  // Create other necessary accounts
  console.log(`\n- Creating accounts...`);
  const initBalance = 100;
  const treasuryKey = PrivateKey.generateED25519();
  const [treasuryAccSt, treasuryId] = await accountCreatorFcn(
    treasuryKey,
    initBalance
  );
  console.log(
    - Created Treasury account ${treasuryId} that has a balance of $
    {initBalance} ¢
  );

  const aliceKey = PrivateKey.generateED25519();
  const [aliceAccSt, aliceId] = await accountCreatorFcn(aliceKey, initBalance);
  console.log(
    - Created Alice's account ${aliceId} that has a balance of ${initBalance} ¢
  );

  // Import the compiled contract bytecode
  const contractBytecode = fs.readFileSync("hbarToAndFromContract.bin");

  // Deploy the smart contract on Hedera
  console.log(`\n- Deploying contract...`);
  let gasLimit = 1000000;
```

```

const [contractId, contractAddress] = await contractDeployFcn(
    contractBytecode,
    gasLimit
);
console.log(- The smart contract ID is: ${contractId});
console.log(
    - The smart contract ID in Solidity format is: ${contractAddress}
);

const tokenId = AccountId.fromString("0.0.47931765");
console.log(\n- Token ID (for association with contract later): ${tokenId});
}

main();

{% endtab %}

{% tab title="HbarToAndFromContract.Sol" %}
javascript
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

// Compile with remix for remote imports to work - otherwise keep precompiles
locally
import
"https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/system-
contracts/hedera-token-service/HederaTokenService.sol";
import
"https://github.com/hashgraph/hedera-smart-contracts/tree/main/contracts/system-
contracts/HederaResponseCodes.sol";

contract hbarToAndFromContract is HederaTokenService{
    //=====
    // GETTING HBAR TO THE CONTRACT
    //=====
    receive() external payable {}

    fallback() external payable {}

    function tokenAssociate(address account, address htsToken) payable external
{
    require(msg.value > 20000000000,"Send more HBAR");

    int response = HederaTokenService.associateToken(account, htsToken);
    if (response != HederaResponseCodes.SUCCESS) {
        revert ("Token association failed");
    }
}

    //=====
    // GETTING HBAR FROM THE CONTRACT
    //=====
    function transferHbar(address payable receiverAddress, uint amount) public {
        receiverAddress.transfer(amount);
    }

    function sendHbar(address payable receiverAddress, uint amount) public {
        require(receiverAddress.send(amount), "Failed to send Hbar");
    }

    function callHbar(address payable receiverAddress, uint amount) public {
        (bool sent, ) = receiverAddress.call{value:amount}("");
        require(sent, "Failed to send Hbar");
    }
}

```


Getting HBAR to the Contract

The \receive/fallback\\\ Functions\\

In this scenario, you (Operator) transfer 10 HBAR to the contract by triggering either the receive or fallback functions of the contract. As described in this Solidity by Example page, the receive function is called when msg.data is empty, otherwise the fallback function is called.

In this case, the helper function contractExecuteNoFcn pays HBAR to the contract by using ContractExecuteTransaction() and specifying a .setPayableAmount() without calling any specific contract function - thus triggering fallback. Note from the Solidity code that receive and fallback are external and payable functions.

The helper function contractCallQueryFcn checks the HBAR balance of the contract by calling the getBalance function of the contract - this call is done using ContractCallQuery().

```
javascript
    console.log(
=====
GETTING HBAR TO THE CONTRACT
=====);

    // Transfer HBAR to the contract using .setPayableAmount WITHOUT
specifying a function (fallback/receive triggered)
    let payableAmt = 10;
    console.log(- Caller (Operator) PAYS ${payableAmt} ¢ to contract
(fallback/receive)...);
    const toContractRx = await contractExecuteNoFcn(contractId, gasLimit,
payableAmt);

    // Get contract HBAR balance by calling the getBalance function in the
contract AND/OR using ContractInfoQuery in the SDK
    await contractCallQueryFcn(contractId, gasLimit, "getBalance"); // Outputs
the contract balance in the console
```

Helper Functions:

```
{% tabs %}
{% tab title="ContractExecuteNoFcn" %}
javascript
async function contractExecuteNoFcn(cId, gasLim, amountHbar) {
    const contractExecuteTx = new ContractExecuteTransaction()
        .setContractId(cId)
        .setGas(gasLim)
        .setPayableAmount(amountHbar);
    const contractExecuteSubmit = await contractExecuteTx.execute(client);
    const contractExecuteRx = await contractExecuteSubmit.getReceipt(client);
    return contractExecuteRx;
}

{% endtab %}

{% tab title="ContractCallQueryFcn" %}
javascript
async function contractCallQueryFcn(cId, gasLim, fcnName) {
    const contractQueryTx = new ContractCallQuery()
        .setContractId(cId)
```

```

        .setGas(gasLim)
        .setFunction(fcnName);
    const contractQuerySubmit = await contractQueryTx.execute(client);
    const contractQueryResult = contractQuerySubmit.getUint256(0);
    console.log(- Contract balance (getBalance fcn): ${contractQueryResult 1e-
8} ¢);
}

```

```

{% endtab %}
{% endtabs %}

```

<details>

<summary>Console Output ☒</summary>

=====

GETTING HBAR TO THE CONTRACT

=====

```

    Caller (Operator) PAYS 10 ¢ to contract (fallback/receive)...
    Contract balance (getBalance fcn): 10 ¢

```

</details>

Executing a Payable Function

Now, you (Operator) transfer 21 HBAR to the contract by calling a specific contract function (tokenAssociate) that is payable using the ContractExecuteTransaction() class and specifying a .setPayableAmount(). This is done with the helper function contractExecuteFcn.

In this scenario, contractParamsBuilderFcn is used to build the parameters that will be passed to the contract function – that is, the contract and token IDs which are then converted to Solidity addresses.

From the Solidity code, note that the tokenAssociate function associates the contract to the HTS token from the first step, and requires more than 20 HBAR to execute (just for fun).

```

javascript
    // Transfer HBAR to the contract using .setPayableAmount SPECIFYING a
contract function (tokenAssociate)
    payableAmt = 21;
    gasLimit = 800000;
    console.log(\n- Caller (Operator) PAYS ${payableAmt} ¢ to contract
(payable function)...);
    const Params = await contractParamsBuilderFcn(contractId, [], 2, tokenId);
    const Rx = await contractExecuteFcn(contractId, gasLimit,
"tokenAssociate", Params, payableAmt);

    gasLimit = 50000;
    await contractCallQueryFcn(contractId, gasLimit, "getBalance"); // Outputs
the contract balance in the console

```

```

{% tabs %}
{% tab title="ContractParamsBuilderFcn" %}

```

```

javascript
async function contractParamsBuilderFcn(aId, amountHbar, section, tId) {
    let builtParams = [];
    if (section === 2) {
        builtParams = new ContractFunctionParameters()

```

```

        .addAddress(aId.toSolidityAddress())
        .addAddress(tId.toSolidityAddress());
    } else if (section === 3) {
        builtParams = new ContractFunctionParameters()
            .addAddress(aId.toSolidityAddress())
            .addUint256(amountHbar * 1e8);
    } else {
    }
    return builtParams;
}


{% endtab %}

{% tab title="ContractExecuteFcn" %}
javascript
async function contractExecuteFcn(cId, gasLim, fcnName, params, amountHbar) {
    const contractExecuteTx = new ContractExecuteTransaction()
        .setContractId(cId)
        .setGas(gasLim)
        .setFunction(fcnName, params)
        .setPayableAmount(amountHbar);
    const contractExecuteSubmit = await contractExecuteTx.execute(client);
    const contractExecuteRx = await contractExecuteSubmit.getReceipt(client);
    return contractExecuteRx;
}

{% endtab %}
{% endtabs %}

```

<details>

<summary>Console Output  </summary>

Caller (Operator) PAYS 21 \hbar to contract (payable function)...
 Contract balance (getBalance fcn): 31 \hbar

</details>

Using \TransferTransaction\\\ in the SDK\\

Lastly in this scenario, the Treasury transfers 30 HBAR to the contract using TransferTransaction(). This is done with the helper function hbar2ContractSdkFcn. This scenario is just a quick recap and reminder of Part 1 of the series, so be sure to give that a read for more details.

```

javascript
// Transfer HBAR from the Treasury to the contract deployed using the SDK
let moveAmt = 30;
const transferSdkRx = await hbar2ContractSdkFcn(treasuryId, contractId,
moveAmt, treasuryKey);
console.log(\n- ${moveAmt}  $\hbar$  from Treasury to contract (via SDK): $
{transferSdkRx.status});

    await contractCallQueryFcn(contractId, gasLimit, "getBalance"); // Outputs
the contract balance in the console

```

Helper Functions:

```

javascript
async function hbar2ContractSdkFcn(sender, receiver, amount, pKey) {
    const transferTx = new TransferTransaction()
        .addHbarTransfer(sender, -amount)
        .addHbarTransfer(receiver, amount)


```

```

        .freezeWith(client);
const transferSign = await transferTx.sign(pKey);
const transferSubmit = await transferSign.execute(client);
const transferRx = await transferSubmit.getReceipt(client);
return transferRx;
}

```

<details>

<summary>Console Output </summary>

```

30 ¢ from Treasury to contract (via SDK): SUCCESS
Contract balance (getBalance fcn): 61 ¢

```

</details>

Getting HBAR from the Contract

In this section the contract transfers HBAR to Alice using three different methods: transfer, send, call. Each transfer is of 20 HBAR, so by the end the contract should have 1 HBAR left in its balance.

This tutorial focuses on implementation. For additional background and details of these Solidity methods, check out Solidity by Example and this external article – just remember that on Hedera, the native cryptocurrency transacted is HBAR, not ETH. One thing worth noting from those resources is that call is currently the recommended method to use.

Contract Transfers HBAR to Alice

The helper function contractExecuteFcn executes the transferHbar function of the contract. The helper function contractParamsBuilderFcn now builds the contract function parameters from the receiver ID (Alice's) and the amount of HBAR to be sent. Also note from the previous section that the contract function is executed with a gasLimit of only 50,000 gas.

```

javascript
    console.log(
=====
GETTING HBAR FROM THE CONTRACT
=====);

    payableAmt = 0;
    moveAmt = 20;

    console.log(- Contract TRANSFERS ${moveAmt} ¢ to Alice...);
    const tParams = await contractParamsBuilderFcn(aliceId, moveAmt, 3, []);
    const tRx = await contractExecuteFcn(contractId, gasLimit, "transferHbar",
tParams, payableAmt);

    // Get contract HBAR balance by calling the getBalance function in the
contract AND/OR using ContractInfoQuery in the SDK
    await showContractBalanceFcn(contractId); // Outputs the contract balance
in the console

```

Helper Functions:


```

javascript
async function showContractBalanceFcn(cId) {
    const info = await new

```

```
ContractInfoQuery().setContractId(cId).execute(client);
    console.log(- Contract balance (ContractInfoQuery SDK): $
{info.balance.toString()});
```

<details>

<summary>Console Output 

=====

GETTING HBAR FROM THE CONTRACT

=====

```
Contract TRANSFERS 20 ¢ to Alice...
Contract balance (ContractInfoQuery SDK): 41 ¢
```

</details>


Contract Sends HBAR to Alice

The same helper function from before now executes the sendHbar function of the contract.

```
javascript
    console.log(\n- Contract SENDS ${moveAmt} ¢ to Alice...);
    const sParams = await contractParamsBuilderFcn(aliceId, moveAmt, 3, []);
    const sRx = await contractExecuteFcn(contractId, gasLimit, "sendHbar",
sParams, payableAmt);

    await showContractBalanceFcn(contractId); // Outputs the contract balance
in the console
```

<details>

<summary>Console Output 

```
Contract SENDS 20 ¢ to Alice...
Contract balance (ContractInfoQuery SDK): 21 ¢
```

</details>

Contract Calls HBAR to Alice

Just like above, the helper function contractExecuteFcn executes the sendHbar function of the contract.

Examine the transaction history for the contract and the operator in the mirror node explorer, HashScan. You can also obtain additional information of interest using the mirror node REST API. Additional context for that API is provided in this blog post.

The last step is to join the Hedera Developer Discord!

```
javascript
    console.log(\n- Contract CALLS ${moveAmt} ¢ to Alice...);
    const cParams = await contractParamsBuilderFcn(aliceId, moveAmt, 3, []);
    const cRx = await contractExecuteFcn(contractId, gasLimit, "callHbar",
cParams, payableAmt);


    await showContractBalanceFcn(contractId); // Outputs the contract balance
```

in the console

```
console.log(\n- SEE THE TRANSACTION HISTORY IN HASHSCAN (FOR CONTRACT AND OPERATOR):  
https://hashscan.io/#/testnet/contract/${contractId}  
https://hashscan.io/#/testnet/account/${operatorId});
```

```
console.log(  
=====  
THE END - NOW JOIN: https://hedera.com/discord  
=====\\n);  
}
```

<details>

<summary>Console Output 

Contract CALLS 20 \hbar to Alice...
Contract balance (ContractInfoQuery SDK): 1 \hbar
SEE THE TRANSACTION HISTORY IN HASHSCAN (FOR CONTRACT AND OPERATOR):

<https://hashscan.io/#/testnet/contract/0.0.47938605><https://hashscan.io/#/testnet/account/0.0.2520793>

</details>

Summary


If you run the entire example successfully, your console should look something like:


<figure><figcaption></figcaption></figure>

This tutorial used the Hedera JavaScript SDK. However, you can try this with the other officially supported SDKs for Java and Go.

Congratulations! 🎉 Now you know how to send HBAR to and from a contract on Hedera using both the SDK and Solidity! Feel free to reach out in Discord if you have any questions!

Additional Resources

 Project Repository

 Have a question? Ask on StackOverflow

<p>Writer: Ed, DevRel Engineer</p> <p>https://github.com/ed-marquez > GitHub https://www.linkedin.com/in/ed-marquez/ > LinkedIn </p></td><td align="center"><p>Editor: Krystal, Technical Writer</p><p>GitHub <a</p>	

```
href="https://hashnode.com/@theekrystallee">Hashnode</a></p></td><td><a href="https://github.com/theekrystallee">https://github.com/theekrystallee</a></td></tr></tbody></table>
```

use-native-tokens-as-erc20-tokens.md:

The Power of Native Hedera Tokens as ERC-20 Tokens: A step-by-step guide

In this tutorial, you'll learn how to make Hedera native tokens work like Ethereum's ERC-20 tokens using the Hashio JSON-RPC instance.

```
<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><strong>1.</strong> <a href="use-native-tokens-as-erc20-tokens.md#prerequisites"><strong>PREREQUISITES</strong></a></td><td><a href="use-native-tokens-as-erc20-tokens.md#prerequisites">#prerequisites</a></td></tr><tr><td align="center"><strong>2.</strong> <a href="use-native-tokens-as-erc20-tokens.md#project-setup"><strong>PROJECT SETUP</strong></a></td><td><a href="use-native-tokens-as-erc20-tokens.md#project-setup">#project-setup</a></td></tr><tr><td align="center"><strong>3.</strong> <a href="use-native-tokens-as-erc20-tokens.md#project-contents"><strong>PROJECT CONTENTS</strong></a></td><td><a href="use-native-tokens-as-erc20-tokens.md#project-contents">#project-contents</a></td></tr><tr><td align="center"><strong>4.</strong> <a href="use-native-tokens-as-erc20-tokens.md#running-the-project"><strong>RUNNING THE PROJECT</strong></a></td><td><a href="use-native-tokens-as-erc20-tokens.md#running-the-project">#running-the-project</a></td></tr></tbody></table>
```

Prerequisites

- Basic understanding of TypeScript and Solidity.
- Get a Hedera testnet account.
- Have ts-node installed.

Table of Contents

1. Project Setup
2. Project Configuration
3. Project Contents
4. Running the Project

Project Setup

To make the setup process simple, you'll use a pre-configured token wrapper project from the token-wrapper repository.

Open a terminal window and navigate to your preferred directory where your project will live. Run the following command to clone the repo and install dependencies to your local machine:

```
bash
git clone https://github.com/Swiss-Digital-Assets-Institute/token-wrapper.git
cd token-wrapper
npm install
```


The dotenv package is used to manage environment variables in a separate .env file, which is loaded at runtime. This helps protect sensitive information like your private keys and API secrets, but it's still best practice to add .env to your .gitignore to prevent you from pushing your credentials to GitHub.

Project Configuration

In this step, you will update and configure the Hardhat configuration file that defines tasks, stores Hedera account private key information, and Hashio Testnet RPC URL. First, rename the .env.example file to .env. and update the .env files with the following code.

Environment Variables

The .env file defines environment variables used in the project. The MYACCOUNTID and MYPRIVATEKEY variables contains the ECDSA Account ID and DER Encoded Private Key, respectively for the Hedera Testnet account. The HEXENCODEDPRIVATEKEY variable contains the HEX Encoded Private Key.

The JSONRPCRELAYURL variable contains the HashIO Testnet endpoint URL. This is the JSON-RPC instance that will submit the transactions to the Hedera test network to test, create and deploy your smart contract.

```
{% code title=".env" %}
bash
MYACCOUNTID =
MYPRIVATEKEY =
HEXENCODEDPRIVATEKEY =
JSONRPCRELAYURL = https://testnet.hashio.io/api

{% endcode %}
```

Project Contents

In this step, you'll examine the descriptions of the project contents in your existing project. If you don't need to review the project contents, you can proceed directly to Running the project.

```
{% tabs %}
{% tab title="artifacts/" %}
This directory contains compiled smart contracts used for generating TypeScript bindings with Typechain. We compile these contracts using Hardhat, a versatile Ethereum development tool. You'll find the configuration in hardhat.config.js.
```

```
Think of Hardhat as your all-in-one environment for Ethereum development tasks like compiling, deploying, testing, and debugging.
{% endtab %}
```

```
{% tab title="contracts/" %}
This directory contains the smart contracts that will be compiled and deployed to the Hedera network. This directory contains the smart contracts that will be compiled and deployed to the Hedera network. Let's take a look at the smart contracts in this directory:
```

```
ERC20.sol - This is the ERC20 token contract.
HederaResponseCodes.sol - This is a contract that contains the response codes for the Hedera network.
HederaTokenService.sol - This contract provides the transactions to interact with the tokens created on Hedera.
```

IHederaTokenService.sol - This is the interface for the HederaTokenService contract.

Vault.sol - This contract, when deployed, manages the functionality of a vault and serves as a testing ground to understand how Hedera native tokens interact with the ERC20 contract.

{% endtab %}

{% tab title=" scripts/" %}

In this directory, you'll find two main scripts: ERC20.ts and utils.ts. They play a crucial role in our interaction with smart contracts. Here's how it all unfolds:

We kick things off by setting up our environment and defining the essential variables. Using the Hedera SDK, we create a new fungible token, which gives us a unique tokenId.

To streamline our interaction with smart contracts, we utilize the Typechain class to create instances of contractERC20 and contractVault. We provide contractERC20 with the tokenId's solidity address and the Ethereum provider. Similarly, we create an instance for contractVault using the vaultContractAddress (after deployed) and the provider. This approach ensures that both contracts are seamlessly integrated into our development process.

With our environment ready, we dive into interactions with the contracts. We demonstrate a token transfer operation, moving native tokens created on Hedera from one account to another through the ERC20 contract. After that, we check how the balance of the receiving account changes. We do this in two ways: first, by calling a function in the SDK, and second, by using the balanceOf function in the ERC20 contract.

In our second example, we deposit 1000 tokens into the Vault contract and then withdraw them. We keep an eye on how this affects the Vault contract's balance. It's worth noting that in Hedera, you need to associate the recipient account with the token before making transfers. To handle this requirement, we use the associate function from the HederaTokenService contract, which establishes the connection between the Vault contract and the token. Once associated, we can easily deposit and withdraw tokens.

{% code title="ERC20.ts" %}

```
typescript
console.clear();
import { AccountId, PrivateKey, TokenAssociateTransaction, Client,
AccountBalanceQuery } from "@hashgraph/sdk";
import { ethers } from "ethers";
import { createFungibleToken, createAccount, deployContract } from "../utils";
```

```
import { ERC20factory } from '../typechain-types/factories/ERC20factory';
import { Vaultfactory } from '../typechain-types/factories/Vaultfactory';
```

```
import { config } from "dotenv";
```

```
config();
```

```
// Provider to connect to the Ethereum network
const provider = new
ethers.providers.JsonRpcProvider(process.env.JSONRPCRELAYURL!);
```

```
// Wallet to sign the transactions
const wallet = new ethers.Wallet(process.env.HEXENCODEDPRIVATEKEY!, provider);
```

```
// Client to interact with the Hedera network
const client = Client.forTestnet();
const operatorPrKey =
PrivateKey.fromStringECDSA(process.env.HEXENCODEDPRIVATEKEY!);
```

```

const operatorAccountId = AccountId.fromString(process.env.MYACCOUNTID!);
client.setOperator(operatorAccountId, operatorPrKey);

async function main() {

    // Create a fungible token with the SDK
    const tokenId = await createFungibleToken("TestToken", "TT",
operatorAccountId, operatorPrKey.publicKey, client, operatorPrKey);

    // Create an account for Alice with 10 hbar using the SDK
    const aliceKey = PrivateKey.generateED25519();
    const aliceAccountId = await createAccount(client, aliceKey, 10);
    console.log(- Alice account id created: ${aliceAccountId!.toString()});

    // Take the address of the tokenId
    const tokenIdAddress = tokenId!.toSolidityAddress();
    console.log(- tokenIdAddress, tokenIdAddress);

    // We connect to the ERC20 contract using typechain
    const account = wallet.connect(provider);
    const accountAddress = account.address;
    console.log(- accountAddress, accountAddress);

    const contractERC20 = ERC20factory.connect(
        tokenIdAddress,
        account
    );

    // We deploy the Vault contract using the SDK and take the address
    const contractVaultId = await deployContract(client, Vaultfactory.bytecode,
4000000);
    const contractVaultAddress = contractVaultId!.toSolidityAddress();
    console.log(- contractVaultId, contractVaultId!.toString());

    // We connect to the Vault contract using typechain
    const contractVault = Vaultfactory.connect(
        contractVaultAddress,
        wallet
    );

    // We set Alice as the operator, now she is the one interacting with the
hedera network
    const aliceClient = client.setOperator(aliceAccountId!, aliceKey);

    // We associate the Alice account with the token
    const tokenAssociate = await new TokenAssociateTransaction()
        .setAccountId(aliceAccountId!)
        .setTokenIds([tokenId!])
        .execute(aliceClient);

    const tokenAssociateReceipt = await tokenAssociate.getReceipt(aliceClient);
    console.log(- tokenAssociateReceipt $
{tokenAssociateReceipt.status.toString()});

    const aliceAccountAddress = aliceAccountId!.toSolidityAddress();
    // We transfer 10 tokens to Alice using the ERC20 contract
    const transfer = await contractERC20.transfer(aliceAccountAddress, 10,
{ gasLimit: 1000000 })
    const transferReceiptWait = await transfer.wait();
    console.log(- Transfer, transferReceiptWait);

    // We check the balance tokenId from Alice using the SDK
    const balanceAliceNativeToken = new AccountBalanceQuery()
        .setAccountId(aliceAccountId!)

```

```

    const transactionQuery = await balanceAliceNativeToken.execute(client);
    const balanceTokenSDK = transactionQuery.tokens!.get(tokenId!);
    console.log("- Balance from Alice using the SDK",
balanceTokenSDK.toString());

    // We check the balance tokenId from Alice using the ERC20 contract
    const balanceAliceERC20 = await
contractERC20.balanceOf(aliceAccountAddress);
    console.log("- Balance from Alice using the ERC20",
parseInt(balanceAliceERC20.toString()));

    // We associate the Vault contract with the token so we can transfer tokens
to it
    const associate = await contractVault.associateFungibleToken(tokenIdAddress,
{ gasLimit: 1000000 })
    const associateReceipt = await associate.wait();
    console.log(- Associate, associateReceipt);

    // We deposit 1000 tokens to the Vault contract
    const deposit = await contractERC20.transfer(contractVaultAddress, 1000,
{ gasLimit: 1000000 })
    const depositReceipt = await deposit.wait();
    console.log(- Deposit tokens to the Vault contract, depositReceipt);

    // We check the balance of the Vault contract after the deposit
    let balanceVaultERC20 = await contractERC20.balanceOf(contractVaultAddress);
    console.log(- Balance of the Vault contract before the withdraw,
parseInt(balanceVaultERC20.toString()));

    // We withdraw 100 tokens from the Vault contract
    const withdraw = await contractVault.withdraw(tokenIdAddress, { gasLimit:
1000000 })
    const withdrawReceipt = await withdraw.wait();
    console.log(- Withdraw tokens from the Vault contract, withdrawReceipt);

    // We check again the balance of the Vault contract after the withdraw to
see if it has changed
    balanceVaultERC20 = await contractERC20.balanceOf(contractVaultAddress);
    console.log(- Balance of the Vault contract after the withdraw,
parseInt(balanceVaultERC20.toString()));
}

main();

{% endcode %}
{% endtab %}

{% tab title="typechain-types/" %}
&#x20;This directory contains the typescript bindings generated by typechain.
These bindings are used to interact with the smart contracts.
{% endtab %}
{% endtabs %}

```

Running the project

Now that you have your project set up and configured, we can run it.

To do so, run the following command:

```
bash
```

```
npm run compile
ts-node scripts/ERC20.ts
```

The first command compiles the smart contracts and generates the typescript bindings. The second command runs the ERC20 script.

To see the transactions on the Hedera network, you can use the Hedera Testnet Explorer.

> Note: At the top of the explorer page, remember to switch the network to TESTNET before you search for the transaction.

<details>

```
<summary>console check ☒
```

bash

[illegible]

[illegible]

[illegible]

[illegible]

Congratulations! 🎉 You have successfully learned to use native Hedera tokens as ERC20 tokens. Feel free to reach out if you have any questions:

Twitter LinkedIn
https://twitter.com/luciamunozdev
<p>Editor: Krystal, Technical Writer</p>
GitHub Twitter
https://github.com/theekrystallee

deploy-and-verify-smart-contract.md:

description: >-

Discover how to deploy and automatically verify your Hedera smart contract. Learn how to verify a pre-existing contract and check a contracts verification status.

How to Deploy and Verify a Hedera Smart Contract with Foundry

What you will accomplish


- [] Deploy your smart contract to Hedera Testnet and automatically verify
- [] Verify a pre-existing smart contract on Hedera Testnet
- [] Check the verification status of a smart contract

Prerequisites

Before you begin, you should be familiar with the following:

JavaScript
Solidity
Foundry


<details>

<summary>Also, you should have the following set up on your computer 

- [x] git installed
 - Minimum version: 2.37
 - Recommended: Install Git (Github)
- [x] A code editor or IDE
 - Recommended: VS Code. Install VS Code (Visual Studio)
- [x] NodeJs + npm installed
 - Minimum version of NodeJs: 18
 - Minimum version of npm: 9.5
 - Recommended for Linux & Mac: nvm
 - Recommended for Windows: nvm-windows
- [x] foundry forge and cast installed
 - forge Minimum version: 0.2.0 (3cdee82 2024-02-15T00:19:38.655803000Z)
 - cast Minimum version: 0.2.0 (3cdee82 2024-02-15T00:19:38.543163000Z)

</details>

<details>

<summary>Check your prerequisites set up  </summary>

Open your terminal, and enter the following commands.

shell

```
git --version
code --version
node --version
npm --version
forge --version
cast --version
```

Each of these commands should output some text that includes a version number, for example:

```
text
git --version
git version 2.39.2 (Apple Git-143)

code --version
1.81.1
6c3e3dba23e8fadc360aed75ce363ba185c49794
arm64

node --version
v20.6.1

npm --version
9.8.1

forge --version
0.2.0 (3cdee82 2024-02-15T00:19:38.655803000Z)

cast --version
0.2.0 (3cdee82 2024-02-15T00:19:38.543163000Z)
```

If the output contains text similar to command not found, please install that item.

</details>

Get started

Set up project

To follow along, start with the main branch, which is the default branch of this repository. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

forge manages dependencies by using git submodules. Clone the following project and pass --recurse-submodules to the git clone command to automatically initialize and update the submodule in the repository.

```
shell
git clone --recurse-submodules git@github.com:hedera-dev/foundry-deploy-and-verify-smart-contract.git
```

<details>

<summary>Alternative with git and HTTPS</summary>

If you haven't configured SSH to work with git, you may wish use this command instead:

```
shell
git clone --recurse-submodules https://github.com/hedera-dev/foundry-deploy-and-verify-smart-contract.git
```

</details>

Install the submodule dependencies

In your terminal, enter the projects root directory

```
shell
cd foundry-deploy-and-verify-smart-contract
```

Next, run the following command to install the dependencies

```
shell
forge install
```

Obtain your ECDSA Hex Encoded Private Key

In order to deploy your smart contract you will need access to your ECDSA hex encoded private key. You can find your ECDSA hex encoded private key by logging into Hedera Portal.

If you need to create a Hedera account, follow the create a hedera portal profile faucet tutorial.

```
{% hint style="warning" %}
Keep your private key accessible as it will be needed in the following steps.
{% endhint %}
```

Choose your Testnet RPCURL

Choose one of the options over at [How to Connect to Hedera Networks Over RPC](#)

```
{% hint style="warning" %}
Keep the Testnet RPCURL accessible as it will be needed in the next step.
{% endhint %}
```

Deploy and Automatically Verify Your Contract on Hedera Testnet

In your terminal, replace the value of "HEXEncodedPrivateKey" with your ECDSA account's private key and replace "RPCURL" with a Testnet URL in the command below:

```
shell
forge create --rpc-url "RPCURL" --private-key "HEXEncodedPrivateKey" --verify --verifier sourcify --verifier-url https://server-verify.hashscan.io src/ToDoList.sol:ToDoList
```

<details>

<summary>Example forge create command </summary>

```
shell
forge create --rpc-url https://testnet.hashio.io/api --private-key
```

```
0x348ce564d427a3317b6536bbcff9290d69395b06ed6c486954e971d960fe87ac --verify --
verifier sourcify --verifier-url https://server-verify.hashscan.io
src/ToDoList.sol:ToDoList
```

</details>

{% hint style="info" %}

Sourcify is a Solidity source code and metadata verification tool.

Hashscan is a Hedera Mirror Node Explorer that integrates with Sourcify to provide a verification service located at <https://server-verify.hashscan.io>.

Learn more

{% endhint %}

You should see output similar to the following:

text

```
[..] Compiling...
```

```
[.] Compiling 22 files with 0.8.23
```

```
[:] Solc 0.8.23 finished in 3.43s
```

```
Compiler run successful!
```

```
Deployer: 0xdfAb7899aFaBd146732c84eD83250889C40d6A00
```

```
Deployed to: 0x3b096B1c56A48119CB4fe140F1D26196590aF46C
```

```
Transaction hash:
```

```
0x32f6a03b569a934d8d1e20bb29a20fe1008f0b3bf9ab41e1f26ed88bb28b3c05
```

```
Starting contract verification...
```

```
Waiting for sourcify to detect contract deployment...
```

```
Start verifying contract 0x3b096B1c56A48119CB4fe140F1D26196590aF46C deployed on
296
```

```
Submitting verification for [ToDoList]
```

```
"0x3b096B1c56A48119CB4fe140F1D26196590aF46C".
```

```
Contract successfully verified
```

Open the Hashscan explorer in your browser by copying the Deployed to EVM address and replacing <DeployedContractEVMAddress> in the link below:

text

<https://hashscan.io/testnet/contract/<DeployedContractEVMAddress>>

{% hint style="success" %}

Example:

<https://hashscan.io/testnet/contract/0x3b096B1c56A48119CB4fe140F1D26196590aF46C>

{% endhint %}

<figure><figcaption><p>Hashscan shows verified smart

contract</p></figcaption></figure>

You should see a page with:

The title "Contract" (1)

An "EVM Address" field that matches the value of Deployed To in the output above. (2)

A section titled "Contract Bytecode" with a green verified tag. (3)

Two tabs titled "Source" and "Bytecode". (4)

Verify A Pre-Existing Contract on Testnet

You may have a pre-existing contract that has not been verified yet.

```
{% hint style="info" %}
```

Deploy a contract without verifying by running this command in your terminal:

```
shell
forge create --rpc-url "RPCURL" --private-key "HEXEncodedPrivateKey"
src/ToDoList.sol:ToDoList
```

```
{% endhint %}
```

In order to verify a pre-existing smart contract you will need:

- the contract EVM address with 0x prefix
- the contract name or path to the contract
- the chain ID

In your terminal, replace <CONTRACTADDRESS> with the contract EVM address you wish to verify.

```
shell
forge verify-contract --chain-id 296 --verifier sourcify --verifier-url
https://server-verify.hashscan.io <CONTRACTADDRESS> src/ToDoList.sol:ToDoList
```

```
{% hint style="info" %}
```

- The chain ID for Mainnet is 295.
- The chain ID for Testnet is 296, which is what you are using.
- The chain ID for Previewnet is 297.

```
{% endhint %}
```

You should see output similar to the following:

```
text
Start verifying contract 0xC08d3Cf01739C713BaD1cf65FD4127CB90550568 deployed on
296
```

```
Submitting verification for [ToDoList]
"0xC08d3Cf01739C713BaD1cf65FD4127CB90550568".
Contract successfully verified
```

Check Contract Verification Status

In your terminal, replace <CONTRACTADDRESS> with the contract EVM address you wish to verify.

```
shell
forge verify-check --chain-id 296 --verifier sourcify --verifier-url
https://server-verify.hashscan.io <CONTRACTADDRESS>
```

You should see output similar to the following:

```
text
Checking verification status on 296
Contract successfully verified
```

Complete

Congratulations, on completing the tutorial on how to verify smart contracts on Hedera Testnet.

You have learned how to:

- [x] Deploy your smart contract to Hedera Testnet and Automatically Verify
- [x] Verify a pre-existing smart contract on Hedera Testnet
- [x] Check a smart contracts verification status

```
<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p><a href="https://github.com/a-ridley">GitHub</a> | <a href="https://linkedin.com/in/abixcastro">LinkedIn</a></p></td><td align="center"><a href="https://linkedin.com/in/abixcastro">https://linkedin.com/in/abixcastro</a></td></tr></tbody></table>
```

fork-hedera-testnet-on-latest-block.md:

description: >-

Forking lets you interact with contracts and run tests as if on a real network. Learn how to fork Hedera Testnet on the latest block and test your contracts with the latest state of the network.

Fork Hedera Testnet: Interact with Deployed Contracts on Latest Block

What you will accomplish


- [] Deploy your smart contract to Hedera Testnet using forge create
- [] Use cast send to sign and publish a transaction that changes the state.
- [] Use cast call to perform a call without changing state.
- [] Fork Hedera Testnet on the latest block & run your tests against your deployed contract

Prerequisites

Before you begin, you should be familiar with the following:

JavaScript
Solidity
Foundry

<details>


<summary>Also, you should have the following set up on your computer 

</summary>

- [x] git installed
 - Minimum version: 2.37
 - Recommended: Install Git (Github)
- [x] A code editor or IDE
 - Recommended: VS Code. Install VS Code (Visual Studio)
- [x] NodeJs + npm installed
 - Minimum version of NodeJs: 18
 - Minimum version of npm: 9.5
 - Recommended for Linux & Mac: nvm
 - Recommended for Windows: nvm-windows
- [x] foundry forge and cast installed
 - forge Minimum version: 0.2.0 (3cdee82 2024-02-15T00:19:38.655803000Z)
 - cast Minimum version: 0.2.0 (3cdee82 2024-02-15T00:19:38.543163000Z)

</details>

<details>

<summary>Check your prerequisites set up  </summary>

Open your terminal, and enter the following commands.

```
shell
git --version
code --version
node --version
npm --version
forge --version
cast --version
```

Each of these commands should output some text that includes a version number, for example:

```
text
git --version
git version 2.39.2 (Apple Git-143)

code --version
1.81.1
6c3e3dba23e8fadc360aed75ce363ba185c49794
arm64

node --version
v20.6.1

npm --version
9.8.1

forge --version
0.2.0 (3cdee82 2024-02-15T00:19:38.655803000Z)

cast --version
0.2.0 (3cdee82 2024-02-15T00:19:38.543163000Z)
```

If the output contains text similar to command not found, please install that item.

</details>

Get started

Set up project

To follow along, start with the main branch, which is the default branch of this repository. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

forge manages dependencies by using git submodules. Clone the following project and pass --recurse-submodules to the git clone command to automatically initialize and update the submodule in the repository.

shell

```
git clone --recurse-submodules git@github.com:hedera-dev/fork-hedera-testnet.git
```

Install the submodule dependencies

```
shell
forge install
```

Copy Your ECDSA Hex Encoded Private Key

Grab your ECDSA hex encoded private key by logging into Hedera Portal.

```
{% hint style="info" %}
If you need to create a Hedera account, follow the create a hedera portal
profile faucet tutorial.
{% endhint %}
```

Deploy your contract to Hedera Testnet

In your terminal, replace the value of "HEXEncodedPrivateKey" with your ECDSA account's private key in the command below:

Next, Replace "RPCURL" with a Tesnet URL by choosing one of the options over at [How to Connect to Hedera Networks Over RPC](#)

```
shell
forge create --rpc-url "RPCURL" --private-key "HEXEncodedPrivateKey"
src/ToDoList.sol:ToDoList
```

```
{% hint style="warning" %} Your hex encoded private key must be prefixed with
0x. {% endhint %}
```

You should see output similar to the following:

```
text
[..] Compiling...
[.] Compiling 22 files with 0.8.23
[.] Solc 0.8.23 finished in 3.44s
Compiler run successful!
No files changed, compilation skipped
Deployer: 0xdfAb7899aFaBd146732c84eD83250889C40d6A00
Deployed to: 0xc1E551Eb1B3430A8D373C43e8804561fca5ce90D
Transaction hash:
0x8709443db7b60df7b563c83514ce8b03e54c341a5fe9844e01c72b05fc50950e
```

Open the Hashscan link to your deployed contract by copying the Deployed to EVM address and replacing <DeployedContractEVMAddress> in the link below:

```
text
https://hashscan.io/testnet/contract/<DeployedContractEVMAddress>
```

```
{% hint style="success" %}
Example:
https://hashscan.io/testnet/contract/0xc1E551Eb1B3430A8D373C43e8804561fca5ce90D
{% endhint %}
```

Execute a contract call and create a new todo

Use Foundry's `cast send` command to sign and publish a transaction to Hedera Tesnet.

Replace "deployed-contract-EVM-address" with your deployed contracts EVM address

Replace "RPCURL" with a Tesnet URL.

```
cast send "deployed-contract-EVM-address" --private-key "HEXEncodedPrivateKey"
"createTodo(string)(uint256)" "Buy camping supplies" --rpc-url "RPCURL"
```

text

```
0x958f246cc010aa074c81eae1988abc16c16bc12d2246397320d163b5ea748a89
```

contractAddress	0xc1E551Eb1B3430A8D373C43e8804561fca5ce90D
-----------------	--

```
effectiveGasPrice      10500000000000
```

gasUsed	91838
---------	-------

```
logs []
```

logsBloom

[illegible][illegible][illegible][illegible][illegible][illegible]

000

root

```
status      1
```

transactionHash

0x18286c79ae735dd08fccbe676ee2d7335147d7a1a6c7f2d0e99099c2802c397c

```
transactionIndex      2
```

type	2
------	---

Use cast call to execute `TodoList.sol`'s `numberOfTodos()`.

shell

```
cast call "deployed-contract-EVM-address" "numberOfTodos()(uint256)" --rpc-url
"RPCURL"
```

You should see `numberOfTodos` has incremented by 1.

```
{% hint style="warning" %}
```

cast send signs/publishes a transaction and changes the state.

cast call performs a call without changing state. Essentially a read transaction.

```
{% endhint %}
```

Write your test

An almost-complete test has already been prepared for you. It's located at `test/ToDoList.t.sol`.
You will only need to make a few modifications (outlined below) for it to run successfully.

```
{% hint style="warning" %}
```

Look for a comment in the code to locate the specific lines of code which you will need to edit. For example, in this step, look for this:

```
// Step (1) in the accompanying tutorial
```

You will need to delete the inline comment that looks like this: `/ ... /`.
Replace it with the correct code.

```
{% endhint %}
```

Step 1: Target your deployed contract

Open the `ToDoList.t.sol` file and copy and paste the line below.

Replace `"DeployedContractEVMAddress"` with your deployed contract's EVM address.

```
solidity
    ToDoList public todoList =
        ToDoList("DeployedContractEVMAddress");
```

Step 2: Perform an assertion

This test saves the current number of todos as `numberOfTodos`, then executes `createTodo()`, and finally performs an assertion to check that the new number of todos has increased.

Write a statement which asserts that the `todoCountAfterCreate` is equal to the `numberOfTodos + 1`.

Copy the below line and paste it in `ToDoList.t.sol`

```
solidity
assertEq(todoCountAfterCreate, (numberOfTodos + 1));
```

Fork test Hedera Testnet and run your test

Using the `--fork-url` flag you will run your test against a forked Hedera Testnet environment at the latest block.

In your terminal, replace `"RPCURL"` with a Testnet URL

```
shell
forge test --fork-url "RPCURL" -vvvv
```

You should see output similar to the following:

```
text
[.] Compiling...
[:] Compiling 1 files with 0.8.23
[.] Solc 0.8.23 finished in 1.15s
Compiler run successful!
```

```
Running 1 test for test/ToDoList.t.sol:ToDoListTest
[PASS] testcreateTodoreturnsNumberOfTodosIncrementedByOne() (gas: 59188)
Traces:
```

```
[59188] ToDoListTest::testcreateTodoreturnsNumberOfTodosIncrementedByOne()
  └─ [2325] 0xc1E551Eb1B3430A8D373C43e8804561fca5ce90D::getNumberOfTodos()
[staticcall]
```

```

    |   L ← 1
    | [51026] 0xc1E551Eb1B3430A8D373C43e8804561fca5ce90D::createTodo("A new
todo for you!")
    |   L ← 2
    |   ← ()

```

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.13s

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)

Your output will show you the state of `numberOfTodos` before you created a new todo and after. It also shows whether the test passed, failed or was skipped.

```
{% hint style="info" %}
```

```
If you'd like to test a contract deployed on mainnet use a Mainnet RPC URL.
Currently fork testing at the latest block is only supported. Be aware everytime
you run your test it is against the latest state of the network.
{% endhint %}
```

Complete

Congratulations, on completing the tutorial on how to fork Hedera Testnet on the latest block.

You have learned how to:

- [x] Deploy your smart contract to Hedera Testnet using forge create
- [x] Use cast send to sign and publish a transaction that changes the state.
- [x] Use cast call to perform a call without changing state.
- [x] Fork Hedera Testnet on the latest block & run your tests against your deployed contract

data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-bbox="450 100 888 125" data-hidden data-card-target data-type="content-ref"></th></tr></thead><tbody><tr><td align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p>GitHub LinkedIn</p></td><td align="center"><p>Editor: Brendan, DevRel Engineer</p><p>GitHub Blog</p></td></tr></tbody></table>

```
# README.md:
```

Foundry

Foundry empowers developers with tools for smart contract development. One of the three main components of Foundry is Forge. Forge is a Foundry command-line tool that allows developers to run tests, build, and deploy smart contracts.

Foundry Key benefits:

Write tests in Solidity & limit your context switching.

EVM cheatcodes give you more control over smart contract development.

This series of mini-tutorials demonstrates how to set up Foundry and use Forge for seamless integration with your Hedera project to test your smart contracts & how to fork Hedera Mainnet to test against deployed contracts.

The tutorials are self-contained and can be done in any order.

```
{% content-ref url="setup-foundry-and-write-basic-unit-test.md" %}
setup-foundry-and-write-basic-unit-test.md
{% endcontent-ref %}
```

```
{% content-ref url="deploy-and-verify-smart-contract.md" %}
deploy-and-verify-smart-contract.md
{% endcontent-ref %}
```

```
{% content-ref url="./test-an-event-with-foundry.md" %}
test-an-event-with-foundry.md
{% endcontent-ref %}
```

```
{% content-ref url="fork-hedera-testnet-on-latest-block.md" %}
fork-hedera-testnet-on-latest-block.md
{% endcontent-ref %}
```

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p><a
href="https://github.com/a-ridley">GitHub</a> | <a
href="https://twitter.com/ridley">Twitter</a></p></td><td><a
href="https://twitter.com/ridley">https://twitter.com/ridley</a></td></tr></
tbody></table>
```

setup-foundry-and-write-basic-unit-test.md:

```
---
description: >-
  Learn how to set up Foundry and use the Forge command-line tool to run your
  smart contract tests written in Solidity.
---
```

How to Setup Foundry and Write a Basic Unit Test

What you will accomplish


- [] Configure Foundry and Forge with a Hedera Project
- [] Write unit tests in Solidity
- [] Run your tests using Foundry forge command
- [] Create a Forge Gas Report

Prerequisites

Before you begin, you should be familiar with the following:

JavaScript
Solidity
Foundry

<details>


<summary>Also, you should have the following set up on your computer 

</summary>

```
[x] git installed
    Minimum version: 2.37
    Recommended: Install Git (Github)
[x] A code editor or IDE
    Recommended: VS Code. Install VS Code (Visual Studio)
[x] NodeJs + npm installed
    Minimum version of NodeJs: 18
    Minimum version of npm: 9.5
    Recommended for Linux & Mac: nvm
    Recommended for Windows: nvm-windows
```

</details>

<details>

<summary>Check your prerequisites set up  </summary>

Open your terminal, and enter the following commands.

```
shell
git --version
code --version
node --version
npm --version
```

Ensure these versions meet or exceed the minimum requirements:

```
git --version
git version 2.39.2 (Apple Git-143)

code --version
1.81.1
6c3e3dba23e8fadc360aed75ce363ba185c49794
arm64

node --version
v20.6.1

npm --version
9.8.1
```

</details>

Get started

Set up project

To follow along, start with the main branch, which is the default branch of this repository. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

```
shell
git clone git@github.com:hedera-dev/setup-foundry-and-write-basic-unit-test.git
```

Add a submodule

Forge manages dependencies by using git submodules. Run the steps below to add and install the git submodules necessary to use Forge.

Next, add the Forge Standard Library to your project:

```
shell
cd setup-foundry-and-write-basic-unit-test
```

```
shell
git submodule add https://github.com/foundry-rs/forge-std lib/forge-std
```

This command will add the forge standard library to our project by creating a folder named lib. The forge standard library is the preferred testing library when working with Foundry.

Install Foundryup

Foundryup represents Foundry's tool management approach. Executing this command will install forge and other essential Foundry CLI tools.

```
shell
curl -L https://foundry.paradigm.xyz | bash
```

You should see output similar to the following:

```
shell
% Total      % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--    0
100 1942 100 1942    0     0 3336      0 --:--:-- --:--:-- --:--:--    0
Installing foundryup...
100.0%
```

Detected your preferred shell is zsh and added foundryup to PATH. Run 'source /Users/abi/.zshenv' or start a new terminal session to use foundryup. Then, simply run 'foundryup' to install Foundry.

```
{% hint style="warning" %}
```

You may need to add foundry to PATH and open a new terminal to make the foundryup command available. Then run foundryup to install Foundry.

```
{% endhint %}
```

<details>

<summary>Once you've installed foundry you should get a similar output</summary>

```
x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x
FOUNDRY Portable and modular toolkit
          for Ethereum Application Development
          written in Rust.
.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x
Repo      : https://github.com/foundry-rs/
Book      : https://book.getfoundry.sh/
Chat      : https://t.me/foundryrs/
Support   : https://t.me/foundrysupport/
Contribute: https://github.com/orgs/foundry-rs/projects/2/
```

```
.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x.x0x
foundryup: installing foundry (version nightly, tag nightly)
foundryup: downloading latest forge, cast, anvil, and chisel
100.0%
foundryup: downloading manpages
100.0%
foundryup: installed - forge 0.2.0 (5ea2c5e 2024-01-22T00:24:09.322705000Z)
foundryup: installed - cast 0.2.0 (5ea2c5e 2024-01-22T00:24:09.341247000Z)
foundryup: installed - anvil 0.2.0 (5ea2c5e 2024-01-22T00:24:09.359481000Z)
foundryup: installed - chisel 0.2.0 (5ea2c5e 2024-01-22T00:24:09.377345000Z)
foundryup: done!
```

</details>

Install the submodule dependencies

```
shell
forge install
```

Remap dependencies

In order to make the import of the forge standard library easier to write, we will remap the dependency.

Open the project setup-foundry-and-write-basic-unit-test, in a code editor.

Create a new text file under the root directory named remappings.txt

Paste in the following line of code

```
forge-std/=lib/forge-std/src/
```

When we want to import from forge-std we will write: import
"forge-std/Contract.sol"

Setup the test

A test file named TodoList.t.sol has been provided to you under the test folder.

On line 7, we see our TodoListTest contract inherits Forge Standard Library's Test contract, which provides us access to the necessary functionality to test our smart contracts.

```
<figure><figcaption><p>TodoList Test
Contract</p></figcaption></figure>
```

Step 1: Create your test instance

Create an instance of the contract TodoList.sol in TodoList.t.sol order to be able to test it.

```
solidity
TodoList public todoList;
```

```
{% hint style="warning" %}
```

Look for a comment in the code to locate the specific lines of code that you will need to edit. For example, in this step, look for this: // Step (1) in the accompanying tutorial. You will need to delete the inline comment that looks like this: /\ ... \/. Replace it with the correct code.

```
{% endhint %}
```

Write a test

Step 2: Deploy a new contract every time you run a test

The `setUp()` function is invoked before each test case is run and is optional. Have the `TodoList.t.sol` test contract deploy a new `TodoList` contract by adding the following code in the `setUp()` function.

```
solidity
todoList = new TodoList();
```

Step 3: Confirm that the number of todos increases by one after calling `createTodo()`

Assert that the `numberOfTodosAfter` executing `createTodo()` is equal to the `numberOfTodosBefore + 1`.

```
solidity
    assertEquals(numberOfTodosAfter, (numberOfTodosBefore + 1), "create todo
test");
```

Build and run your test

Foundry expects the test keyword as a prefix to distinguish a test. Therefore, all tests you want to run must be prefixed with the test keyword.

Foundry expects the test keyword as a prefix to distinguish a test. Therefore, all tests that you want to run must be prefixed with the test keyword.

In the terminal, ensure you are in the root project directory and build the project.

```
shell
forge build
```

You should see output similar to the following:

```
text
[..] Compiling...
[.] Compiling 22 files with 0.8.23
[.] Solc 0.8.23 finished in 3.44s
Compiler run successful!
```

After a successful build, run your test.

```
shell
forge test
```


You should see output similar to the following:

```
text
[.] Compiling...
No files changed, compilation skipped

Running 1 test for test/ToDoList.t.sol:ToDoListTest
[PASS] testcreateTodoreturnsNumberOfTodosIncrementedByOne() (gas: 76346)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.12ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

By default forge test only displays a minimal summary of a test, whether it failed or passed. You can display more detailed information by using the -v flag and increasing the verbosity.

In the terminal, re-run your test but include a verbosity level 4. This will display stack traces for all tests, including the setup.

```
shell
forge test -vvvv

<details>

<summary>Level 4 verbosity output</summary>

<details>

<summary>Level 4 verbosity output</summary>
```

You should see output similar to the following:

```
text
[.] Compiling...
No files changed, compilation skipped

Running 1 test for test/ToDoList.t.sol:ToDoListTest
[PASS] testcreateTodoreturnsNumberOfTodosIncrementedByOne() (gas: 76346)
Traces:
  [76346] ToDoListTest::testcreateTodoreturnsNumberOfTodosIncrementedByOne()
    └─ [2325] ToDoList::getNumberOfTodos() [staticcall]
      └─ ── 0
    └─ [68126] ToDoList::createTodo("A new todo for you!")
      └─ ── 1
    └─ ── ()
```

```
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 580.08µs

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

</details>

</details>

Forge Gas Reports

Forge has functionality built in to give you gas reports of your contracts. You can specify which contract should generate a gas report in the foundry.toml file.

The foundry.toml file is a configuration file that is used to configure forge.

Create a new file in the root directory named foundry.toml. Paste the following contents.

```
toml
[profile.default]
src = 'src'
out = 'out'
libs = ['lib']
```

Step 4 - Configure foundry to produce a gas report for TodoList.sol
#/ ... /

```
[rpcendpoints]
htestnet = "https://testnet.hashio.io/api"
hmainnet = "https://mainnet.hashio.io/api"
```

See more config options <https://github.com/foundry-rs/foundry/tree/master/config>

Step 4: Configure foundry to produce a gas report for TodoList.sol

Replace the comment `#/ ... /` with the line below:

```
toml
gasreports = ["TodoList"]
```

In the terminal, generate a gas report.

```
shell
forge test --gas-report
```

You should see output similar to the following:

```
<figure><figcaption><p>Test Contract Gas Report</p></figcaption></figure>
```

Your output will show you an estimated gas average, median, and max for each contract function used in a test and total deployment cost and size.

Complete

Congratulations, you have completed how to setup Foundry and write a basic unit test.

You have learned how to:

- [x] Configure Foundry and forge with a hedera project
- [x] Write unit tests in Solidity
- [x] Run your tests using Foundry forge command
- [x] Create a forge gas report

```
<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p><a
href="https://github.com/a-ridley">GitHub</a> | <a
href="https://twitter.com/ridley">Twitter</a></p></td><td><a
```

```
href="https://twitter.com/ridley">https://twitter.com/ridley</a></td></tr><tr><td align="center"><p>Editor: Brendan, DevRel Engineer</p><p><a href="https://github.com/bguiz">GitHub</a> | <a href="https://blog.bguiz.com">Blog</a></p></td><td><a href="https://blog.bguiz.com">https://blog.bguiz.com</a></td></tr></tbody></table>
```

test-an-event-with-foundry.md:

description: >-

In this tutorial, you'll learn how to use Foundry's 'cheatcodes'—special commands that allow you to test and manipulate blockchain states. We'll focus on the vm.expectEmit cheatcode to test events.

How to Test a Solidity Event with Foundry

What you will accomplish


- [] Use the vm.expectEmit Cheatcode
- [] Test a Solidity event

Prerequisites

Before you begin, you should be familiar with the following:

- JavaScript
- Solidity
- Foundry

<details>


<summary>Also, you should have the following set up on your computer 

</summary>

- [x] git installed
 - Minimum version: 2.37
 - Recommended: Install Git (Github)
- [x] A code editor or IDE
 - Recommended: VS Code. Install VS Code (Visual Studio)
- [x] NodeJs + npm installed
 - Minimum version of NodeJs: 18
 - Minimum version of npm: 9.5
 - Recommended for Linux & Mac: nvm
 - Recommended for Windows: nvm-windows
- [x] foundry forge and cast installed
 - forge Minimum version: 0.2.0
 - cast Minimum version: 0.2.0

</details>

<details>

<summary>Check your prerequisites set up 

</summary>

Open your terminal, and enter the following commands.

```
shell
git --version
```

```
code --version
node --version
npm --version
forge --version
cast --version
```

Each of these commands should output some text that includes a version number, for example:

```
text
git --version
git version 2.39.2 (Apple Git-143)

code --version
1.81.1
6c3e3dba23e8fadc360aed75ce363ba185c49794
arm64

node --version
v20.6.1

npm --version
9.8.1

forge --version
0.2.0 (6fcbdd8 2023-12-15T00:29:51.472038000Z)

cast --version
0.2.0 (6fcbdd8 2023-12-15T00:29:51.851258000Z)
```

If the output contains text similar to command not found, please install that item.

</details>

Get started

Set up project

To follow along, start with the main branch, which is the default branch of this repository. This gives you the initial state from which you can follow along with the steps as described in the tutorial.

```
{% hint style="warning" %}
Learn how to setup foundry by completing the Setup Foundry and Write a Basic
Unit Test tutorial. This tutorial will not walkthrough setting up Foundry.
{% endhint %}
```

forge manages dependencies by using git submodules. Clone the following project and pass --recurse-submodules to the git clone command to automatically initialize and update the submodule in the repository.

```
shell
git clone --recurse-submodules git@github.com:hedera-dev/test-an-event-with-
foundry.git
```

Install the submodule dependencies

```
shell
forge install
```

Open the project `test-an-event-with-foundry`, in a code editor.

If you completed the previous tutorial, you may notice the contents of the contract `ToDoList.sol` have changed. Specifically, there is a `CreateTodo` event that has been declared and is emitted in the `createTodo()` function.

Write the test

An almost-complete test has already been prepared for you. It's located at `test/ToDoList.t.sol`.

You will only need to make a few modifications (outlined below) for it to run successfully.

```
{% hint style="warning" %}
```

Look for a comment in the code to locate the specific lines of code which you will need to edit. For example, in this step, look for this:

```
// Step (1) in the accompanying tutorial
```

You will need to delete the inline comment that looks like this: `/ ... /`.

Replace it with the correct code.

```
{% endhint %}
```

Step 1: Define the expected event

Declare the `CreateTodo` event in your test contract. This event is identical to the one that is declared in `ToDoList.sol`.

```
solidity
event CreateTodo(address indexed creator, uint256 indexed todoIndex, string
description);
```

Step 2: Get the current number of todos

Grab the number of todos, as it will be used to create the expected event according to the current state of the contract.

```
solidity
uint256 numberOfTodosBefore = todoList.getNumberOfTodos();
```

Step 3: Specify the event data to test

The cheatcode `vm.expectEmit()` will be used to check if the event is emitted. This cheatcode expects four inputs:

```
bool checkTopic1 asserts the first index
bool checkTopic2 asserts the second index
bool checkTopic3 asserts the data for index 3
bool checkData asserts the remaining data emitted by the event
address emitter asserts the emitting address matches
```

The event being tested includes two indexed arguments: `address indexed creator` and `uint256 indexed todoIndex`. Therefore, we want to assert the matching of topic 1, topic 2, the non-indexed data, and the emitting address with our actual event.

```
solidity
vm.expectEmit(true, true, false, true, address(todoList));
```

Step 4: Emit the expected event

Emit the expected event with the following parameters:

- the creator of the todo is this test contract with address 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496,
- the todoIndex is the current numberOfTodos + 1,
- the description is set to "a new todo."

```
solidity
emit CreateTodo(address(this), numberOfTodosBefore + 1, 'a new todo');
```

Step 5: Execute the contract function that emits the event

Execute TodoList.sol's createTodo() function.

```
solidity
todoList.createTodo(address(this), 'a new todo');
```

Run the test

Step 6: Execute the test

```
shell
forge test --match-test testemitcreateTodoEvent -vvvv
```

You should see output similar to the following:

```
text
[.] Compiling...
No files changed, compilation skipped
```

Running 1 test for test/ToDoList.t.sol:ToDoListTest

[PASS] testemitcreateTodoEvent() (gas: 84576)

Traces:

```
[84576] ToDoListTest::testemitcreateTodoEvent()
├─ [2325] ToDoList::getNumberOfTodos() [staticcall]
│   └─ 0
├─ [0] VM::expectEmit(true, true, false, true, ToDoList:
[0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f])
│   └─ ()
├─ emit CreateTodo(creator: ToDoListTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], todoIndex: 1, description: "a new
todo")
├─ [70920] ToDoList::createTodo(ToDoListTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], "a new todo")
│   └─ emit CreateTodo(creator: ToDoListTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], todoIndex: 1, description: "a new
todo")
│       └─ 1
└─ ()
```

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.10ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)

The test passed and shows the event is working as expected.

Complete

Congratulations, you have completed how to test a solidity event using Foundry. You have learned how to:

- [x] Use the vm.expectEmit Cheatcode
- [x] Test a Solidity event

```
<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target data-type="content-ref"></th><tr><td align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p><a href="https://github.com/a-ridley">GitHub</a> | <a href="https://twitter.com/ridley">Twitter</a></p></td><td align="center"><p>Editor: Michael Garber, Developer Advocate</p><p><a href="https://github.com/mgarbs">GitHub</a> | <a href="https://twitter.com/michaelgarber87">Twitter</a></p></td></tr></thead><tbody><tr><td align="center"><p>Writer: Abi Castro, DevRel Engineer</p><p><a href="https://github.com/a-ridley">GitHub</a> | <a href="https://twitter.com/ridley">Twitter</a></p></td><td align="center"><p>Editor: Michael Garber, Developer Advocate</p><p><a href="https://github.com/mgarbs">GitHub</a> | <a href="https://twitter.com/michaelgarber87">Twitter</a></p></td></tr></tbody></table>
```

hardhat.md:

```
---
description: >-
  Hardhat and EthersJs tutorial - HSCS workshop. Learn how to enable custom
  logic & processing on Hedera through smart contracts.
---
```

Hardhat and EthersJs

Video

```
{% embed url="https://www.youtube.com/watch?v=HfoUCRp8Dk" %}
Hedera Smart Contract Service Workshop Part 5/6 | Hardhat & EtherJs
{% endembed %}
```

Hardhat and EthersJs

Hardhat is a development framework, that is designed specifically to enable smart contract development workflows. EthersJs is a software library, which enables client application development in Javascript. These two work very well together as hardhat integrates strongly with EthersJs by augmenting EthersJs with many convenience and utility functions that improve the developer experience of smart contract development.

Prerequisites

- ✓ Complete the Introduction section of this same tutorial.
- ✓ Optionally, complete the Hedera SDK JS section of this tutorial.

Set up the project

To follow along, enter the hederasdkjs directory within the accompanying tutorial GitHub repository, which you should already have cloned in the Intro section earlier.

Then install dependencies from npm.

```
shell
cd ./hardhat
npm install
```

Step J1: Copy smart contract

We have already written the smart contract in the Intro section of this tutorial. Let's copy that into this directory so that we may continue working on it.

```
shell
cp ../intro/trogdor.sol ./contracts/trogdor.sol
```

Hardhat REPL

A Read-Evaluate-Print Loop (REPL) is an environment which takes input from you, executes that input, and prints the result as output; then start over again. The POSIX-compliant shell that you have been using, such as bash or zsh, is an example of this.

Hardhat has its own REPL feature, which executes commands in the context of the smart contract project you are developing, and does so while connected to a specific EVM-compatible network.

Let's fire up the Hardhat REPL, connected to Hedera Testnet.

```
shell
npx hardhat console --network hederatestnet
```

```
{% hint style="info" %}
```

Note that the network name here is hederatestnet, which is defined within the configuration file hardhat.config.js.

```
{% endhint %}
```

You will notice that the prompt prefix changes, and is now > . Whatever commands you enter now are no longer going to be executed by the regular shell, but instead by the Hardhat REPL. You can enter a .exit command to exit the Hardhat REPL, and return to your regular shell at any time. Let's execute a few commands within the Hardhat REPL before we do exit.

Get block number

Hedera is a distributed ledger technology (DLT), however, it is not a blockchain. A blockchain network groups transactions together into blocks, and achieves network consensus on whether or not a block of transactions is valid, and which block (and therefore transactions) is the next one that should be added to the network. Hedera does not do that, instead it using a different consensus algorithm (Hashgraph), which achieves consensus on individual transactions, and adds them to the network as individual transactions, without a grouping of any kind.

That being said, in order to attain interoperability with EVM-compatible networks, the concept of blocks was introduced, in a manner that does not involve the consensus algorithm. In effect it simply deems all transactions successfully added to the network to be part of the same block based on their timestamp, once approximately every 2 seconds.

JSON-RPC is a Remote Procedure Call protocol, where the requests and responses

are serialised in JSON. Importantly, Ethereum has defined a JSON-RPC API, and this API has become the de-facto standard API to interact with EVM-compatible networks.

```
{% hint style="info" %}
Ref: Hedera: What is Hashgraph Consensus
Ref: HIP-415: Introduction of Blocks
Ref: JSON-RPC: Specification
Ref: Ethereum: JSON-RPC API
{% endhint %}
```

Let's verify that we are able to interact with Hedera Testnet using JSON-RPC by issuing an `ethgetBlockByNumber` JSON-RPC request. The expected response will be the most recent block's number on the network.

```
js
(await require('hardhat').network.provider.send('ethgetBlockByNumber',
['latest', false])).number
```

This does indeed respond with a block number, in hexadecimal.

```
'0x6f8741'
```

Check block number on Hashscan

Convert this to decimal: `0x6f8741 --> 7309121`
Check on Hashscan: <https://hashscan.io/testnet/blocks> --> latest block is 7309352 --> more, as new blocks occur every 2 seconds, approximately
Click on it: <https://hashscan.io/testnet/block/7309352> --> timestamp is 3:41:10.0120 PM Jul 14, 2023 --> matches time now

Address

Let's continue with the REPL, and issue another command. This time it will not be a JSON-RPC request, but rather querying Hardhat itself to see which account we'll be using by default when performing any requests.

```
{% hint style="info" %}
The accounts that Hardhat uses are generated from the seed phrase in the .env
file, plus the derivation path, using logic similar to the following code.
```

```
javascript
const seedPhrase = process.env.BIP39SEEDPHRASE;
const accounts = {
  mnemonic: seedPhrase,
  path: "m/44'/60'/0'/0",
};
```

Note that this has already been done for you in `hardhat.config.js`.

```
{% endhint %}
```

```
js
(await hre.ethers.getSigners())[0].address
```

```
{% hint style="info" %}
Here, hre is a global object exported by Hardhat, and it stands for Hardhat
Runtime Environment.
```

The `hre.ethers` object exposes an instance of the EthersJs software library that

has been initialised by Hardhat using the configuration from `hardhat.config.js`. This includes the signers which are a list of several accounts.

```
{% endhint %}
```

This outputs an EVM address of a Hedera EVM account.

```
'0x07ffaADFe3a598b91ee08C88e5924be3EfF35796'
```

If you have completed the Hedera SDK JS section of this tutorial, you will notice that this is different from the account used there, which was `0x7394111093687e9710b7a7aeba3ba0f417c54474`. This is because the script used for the Hedera SDK JS account was configured to use the operator account. Hardhat, on the other hand, uses one of the EVM accounts generated using the BIP-39 seed phrase. These were generated during Step B4: Fund several Hedera EVM accounts in the Intro section of this tutorial.

<details>

<summary>Hardhat accounts</summary>

If you take a look at the EVM addresses generated by the script - this was its filtered output:

```
#0 EVM address: 07ffaadfe3a598b91ee08c88e5924be3eff35796
#1 EVM address: 1c29e31d241f0d06f3763221f5224a6b82f09cce
```

If you run the signer address query 2 times, changing only the index with each query, you will get exact matches.

```
javascript
> (await hre.ethers.getSigners())[0].address
'0x07ffaADFe3a598b91ee08C88e5924be3EfF35796'
> (await hre.ethers.getSigners())[1].address
'0x1C29e31D241F0D06F3763221F5224A6b82f09Cce'
```

</details>

Alright, you've completed the checks needed, verifying that you are able to successfully connected to Hedera Testnet.

Exit the REPL.

```
js
.exit
```

You'll now return to your regular shell.

Check address on Hashscan

Copy the EVM address that Hardhat uses by default
Go to Hashscan, and search for that address
You should get redirected to an Account page
For example, from:
<https://hashscan.io/testnet/account/0x07ffaADFe3a598b91ee08C88e5924be3EfF35796>
to: <https://hashscan.io/testnet/account/0.0.3996359>
This verifies that the account exists
Check that the account has a balance of HBAR
If it does not exist, or does not have balance, you'll need to create or fund

it before proceeding

To do so, you'll need to repeat Step B4: Fund several Hedera EVM accounts from the Setup section of this tutorial.

Compiling smart contracts

Using hardhat to compile smart contracts

In the Hedera SDK JS section of this tutorial, in the Using solc to compile smart contracts step, you manually installed a specific version of the Solidity compiler, and then ran that in the shell.

Hardhat takes care of that for you, it will simply read which version of the Solidity compiler has been set in the configuration in `hardhat.config.js`, and then install that particular version if necessary, and run it, and manage its outputs including caching where relevant.

```
shell
```

```
npx hardhat compile
```

Let's take a look at the compiled outputs, and where Hardhat stores them.

```
Build cache:hardhat/cache/solidity-files-cache.json
ABI: hardhat/artifacts/contracts/trogdor.sol/Trogdor.json
Bytecode + other compiler outputs: hardhat/artifacts/build-info/${SOMEHASH}.json
```

You do not need to do anything with these, just good to know what is happening behind the scenes.

HAPIs and EVM transactions

However, Hardhat + EthersJs do not understand HAPIs, and are only aware of the EVM transaction model. Thankfully, Hedera supports a HAPI named `EthereumTransaction`, defined in HIP-410. All the different types of EVM transactions are supported through this single HAPI.

This is the low-level protocol supported by the Hedera network which is the gateway for software libraries, developer tools, developer frameworks, and even end use software such as wallets, which were originally designed to work with Ethereum, to also work on Hedera, and is an integral part of Hedera's EVM-compatibility.

```
{% hint style="info" %}
  Ref: HIP-410 - Wrapping Ethereum Transaction Bytes in a Hedera Transaction
{% endhint %}
```

In this section of the tutorial, you are using Hardhat and EthersJs, and this framework/ software library are unaware of HAPIs, including `ContractCreateTransaction`, `ContractExecuteTransaction`, and `ContractCallQuery`, which you used earlier.

The only API that they are able to use is JSON-RPC, and this is where the `EthereumTransaction` HAPI comes into play. When you send a JSON-RPC request to an RPC endpoint for a Hedera network, that gets converted into an `EthereumTransaction` HAPI. The same happens in reverse with the response.

Deploying smart contracts

Using Hardhat to deploy smart contracts

Let's begin by entering the REPL again.

```
shell
npx hardhat console --network hederatestnet
```

Next, let's use the EthersJs APIs to deploy the smart contract.

```
{% hint style="info" %}
```

Note that this will send an EVM deployment transaction to Hedera Testnet, using the following sequence:

```
ContractFactory.deployTransaction EthersJs Javascript API -->
ethsendRawTransaction JSON-RPC request --> EthereumTransaction HAPI
{% endhint %}
```

A deployment is performed via a transaction, just like any other interaction with the network which may change the state of the network. The transaction needs to be performed by an account, in this case, since you are performing an EthereumTransaction, the account needs to be an EVM account.

```
{% hint style="info" %}
```

EthereumTransactions may only be signed using ECDSA secp256k1 keys, which Hedera EVM accounts use.

Hedera-native accounts such as the operator account, on the other hand, use EdDSA Ed25519 keys, and therefore EthereumTransactions may not be signed by them.

```
{% endhint %}
```

The account signing the transaction needs to pay for the cost of processing the transaction. This is a variable fee known as gas. The account that you'll use for signing is, by default, the first EVM account generated in the script from the Intro section of this tutorial, i.e. the one with the hierarchical derivation path of m/44'/60'/0'/0/0.

<details>

<summary>Gas-related terminology</summary>

On Hedera networks, gas is paid for using HBAR.

Gas is denominated in tinybars, and is equal to gas price multiplied by gas units. Both of these are variable in different ways:

Gas price depends on the demand for, and supply of, computational resources of the nodes in the Hedera network. Gas price usually increases with demand levels, and against supply levels.

Gas units depend on the computational and storage costs as metered by the EVM when processing that specific transaction. When processing a transaction, each bytecode has a specific numeric cost, and a running tally of their sum is tracked during transaction processing.

</details>

Back to the Hardhat REPL, enter the following command to obtain the default signer, which is the first EVM account mentioned above.

```
javascript
deployer = (await hre.ethers.getSigners())[0];
```

This will output a SignerWithAddress object, similar to this:

```
javascript
SignerWithAddress {
```

```

isSigner: true,
address: '0x07ffAaDFe3a598b91ee08C88e5924be3EeF35796',
signer: JsonRpcSigner {
  isSigner: true,
  provider: EthersProviderWrapper {
    / ... truncated ... /
  },
  address: '0x07ffAaDFe3a598b91ee08C88e5924be3EeF35796',
  index: null
},
provider: EthersProviderWrapper {
  / ... truncated ... /
}
}

```

Next, instantiate an instance of ContractFactory. The Hardhat-augmented version of EthersJs is aware of the directory structure, and where to find the Solidity compiler's outputs: The binary file containing the EVM bytecode, and the JSON file containing the ABI.

```

javascript
trogdorFactory = await hre.ethers.getContractFactory('Trogdor');

```

This will output a ContractFactory object, similar to this:

```

javascript
ContractFactory {
  bytecode:
'0x608060405234801561001057600080fd5b5061024f806100206000396000f3fe6080604/...tr
uncated.../f119714d7ccd15a6a111ef132d4f9418c614ca4145164736f6c63430008110033',
  interface: Interface {
    fragments: / ... truncated ... /
    abiCoder: AbiCoder { coerceFunc: null },
    functions: {
      'MINFEE()': [FunctionFragment],
      'amounts(address)': [FunctionFragment],
      'burninate()': [FunctionFragment],
      'totalBurnt()': [FunctionFragment]
    },
    errors: {},
    events: { 'Burnination(address,uint256)': [EventFragment] },
    structs: {},
    deploy: ConstructorFragment / ... truncated ... /,
    isInterface: true
  },
  signer: SignerWithAddress {
    isSigner: true,
    address: '0x07ffAaDFe3a598b91ee08C88e5924be3EeF35796',
    signer: JsonRpcSigner / ... truncated ... /,
    provider: EthersProviderWrapper / ... truncated ... /
  }
}

```

You can observe that it has the EVM bytecode, and has parsed the ABI into an interface.

Next, deploy the smart contract. Note that this transaction includes a network request, and so expect to wait for several seconds - it will not be instantaneous like the previous commands.

```

javascript

```

```
trogdor = await trogdorFactory.deploy();
```

This will output a Contract object, similar to this:

```
javascript
Contract {
  interface: Interface {
    fragments: / ... truncated ... /,
    abiCoder: AbiCoder { coerceFunc: null },
    functions: / ... truncated ... /,
    errors: {},
    events: { 'Burnination(address,uint256)': [EventFragment] },
    structs: {},
    deploy: ConstructorFragment / ... truncated ... /,
    isInterface: true
  },
  provider: EthersProviderWrapper / ... truncated ... /,
  signer: SignerWithAddress / ... truncated ... /,
  callStatic: / ... truncated ... /,
  estimateGas: / ... truncated ... /,
  functions: {
    'MINFEE()': [Function (anonymous)],
    'amounts(address)': [Function (anonymous)],
    'burninate()': [Function (anonymous)],
    'totalBurnt()': [Function (anonymous)],
    MINFEE: [Function (anonymous)],
    amounts: [Function (anonymous)],
    burninate: [Function (anonymous)],
    totalBurnt: [Function (anonymous)]
  },
  populateTransaction: / ... truncated ... /,
  filters: {
    'Burnination(address,uint256)': [Function (anonymous)],
    Burnination: [Function (anonymous)]
  },
  runningEvents: {},
  wrappedEmits: {},
  address: '0xeb9922B24D82603A543C764A3e4c9BC451FB8752',
  resolvedAddress: Promise {
    '0xeb9922B24D82603A543C764A3e4c9BC451FB8752',
    [Symbol(asyncid symbol)]: 2916,
    [Symbol(trigger asyncid symbol)]: 2135
  },
  'MINFEE()': [Function (anonymous)],
  'amounts(address)': [Function (anonymous)],
  'burninate()': [Function (anonymous)],
  'totalBurnt()': [Function (anonymous)],
  MINFEE: [Function (anonymous)],
  amounts: [Function (anonymous)],
  burninate: [Function (anonymous)],
  totalBurnt: [Function (anonymous)],
  deployTransaction: {
    hash: '0x0649b37b5cefdb06b2f3139464b1df48498d93af9428906156dfd75831ac0cde',
    type: 2,
    accessList: null,
    blockHash:
'0x910f76e621d2905f6bc2b77ace84adbead9e08c7121b2e1c7bf972965165568b',
    blockNumber: 7623259,
    transactionIndex: 6,
    confirmations: 1,
    from: '0x07ffAaDFe3a598b91ee08C88e5924be3EFf35796',
    maxFeePerGas: BigNumber { value: "194" },
    gasLimit: BigNumber { value: "320000" },
```

```
    to: '0x00000000000000000000000000000000eD3909',
    value: BigNumber { value: "0" },
    nonce: 145,
    data:
'0x608060405234801561001057600080fd5b5061024f806100206000396000f3fe6080604/...truncated.../f119714d7ccd15a6a111ef132d4f9418c614ca4145164736f6c63430008110033',
    r: '0x15f671fe22c9d56fb6725acde2008737549cec491e8c6f703dd308dafd1c7df7',
    s: '0x68c6a4479a71cf783ab8f3fb2bcece0e7ec3441fff298f3c2ff75d80585bcb9a8',
    v: 1,
    creates: null,
    chainId: 296,
    wait: [Function (anonymous)]
  }
}
```

There is much more information on this object, compared to the object from the previous step, because it has been deployed.

Let's examine the deployment transaction for the smart contract in a bit more detail.

```
javascript
trogdorDeployment = await trogdor.deployTransaction.wait();
```

This will output an object, similar to this:

[illegible]

The smart contract has just been deployed!

We will need the `contractAddress` property, so copy that for later use.

Using Hashscan to check smart contract deployment

Copy the output smart contract account address, e.g. 0xeB9922B24D82603A543C764A3e4c9BC451FB8752.
Visit Hashscan

Paste the copied address into the search box
You should get redirected to a "Contract" page, e.g.
<https://hashscan.io/testnet/contract/0.0.15546633>
In it, you can see the EVM address, e.g.
0xeb9922b24d82603a543c764a3e4c9bc451fb8752
Under "Contract Bytecode", you can see "Runtime Bytecode"

Interacting with smart contracts

Using Hardhat to interact with smart contracts

While you still have the Hardhat REPL open, let's continue by interacting with the smart contract that you have just deployed.

Issue a query of the MINFEE constant.

```
javascript
await trogdor.functions.MINFEE();
```

This should reply with a BigNumber object, whose value is 100.

```
javascript
[ BigNumber { value: "100" } ]
```

<details>

<summary><code>BigNumber</code> library vs <code>bigint</code> primitive type</summary>

When EthersJs, and other similar JavaScript libraries such as Web3Js, were originally created, the JavaScript language did not have a means to represent integers greater than 2^{53} .

Therefore, to be able to handle uint256 and other large number types, the BigNumber library was used.

Now this is redundant, because JavaScript has since added a bigint primitive type, which is able to comfortably handle uint256 values. Future versions of EthersJs and Web3Js are likely to deprecate the use of BigNumber.

Ref: MDN - BigInt

</details>

Issue a similar query, this time invoking the totalBurnt function.

```
javascript
await trogdor.functions.totalBurnt();
```

This returns a value of 0, which is to be expected, because we have yet to invoke burninate.

```
javascript
[ BigNumber { value: "0" } ]
```

That is precisely what we'll do next: Invoke burninate. This is a payable function, which expects the transaction to contain a minimum of 100 tinybars sent along with it. Let's sent 123 tinybars.

```
javascript
```



```
await trogdor.functions.burninate({ value: 123n 10000000000n });
```

<details>

<summary>Tinybars vs Weibars</summary>

If you compare the Hedera SDK JS section of this tutorial, to this section of the tutorial, you might notice that there is a difference here.

When using Hedera SDK JS:

```
javascript
.setPayableAmount(new Hbar(123, HbarUnit.Tinybar))
```

When using EthersJs:

```
javascript
{ value: 123n 100000000000n }
```

You might think: Why does the number of tinybar need to be multiplied by 10 million?

The Value of gas price and value fields part of HIP-410 offers this explanation.

For Ethereum transactions, we introduce the concept of "WeiBars", which are 1 to 10^{-18} th the value of a HBAR. This is to maximize compatibility with third party tools that expect ether units to be operated on in fractions of 10^{18} , also known as a Wei. Thus, 1 tinyBar is 10^{10} weiBars or 10 gWei. When calculating gas prices and transferred value the fractional parts below a tiny bar are dropped when converted to tinyBars.

Ref: HIP-410 - Wrapping Ethereum Transaction Bytes in a Hedera Transaction - Value of gas price and value fields

The compatibility above refers to may software libraries, developer tools, developer frameworks, and even end user software such as wallets, assuming/hardcoding the value of the "full unit" in relation to the "fractional unit" of the native currency of that network. Since 1 HBAR = 10^8 tinybar, but 1 Ether = 10^{18} wei, when using client software originally designed to interact with Ethereum, you need to be aware of this and convert between the two.

</details>

This will output some details about the transaction.

```
javascript
{
  hash: '0x12d7e21b0b284a191b962f6fb960b81ba2fcbe68ed5c1b0a560b29ed5dfafbe2',
  type: 2,
  accessList: null,
  blockHash:
'0x6469685f147658e4ca83a3aa9571c5b5bc673ddc03b853b9400e52743a82e3de',
  blockNumber: 7662273,
  transactionIndex: 11,
  confirmations: 1,
  from: '0x07ffaDfE3a598b91ee08C88e5924be3EFf35796',
  maxFeePerGas: BigNumber { value: "197" },
  gasLimit: BigNumber { value: "320000" },
  to: '0x0000000000000000000000000000000000000000eD3909',
  value: BigNumber { value: "123" },
  nonce: 146,
```

```

data: '0x3024480d',
r: '0x7dc8e42fbfed582b8fba03b6df14b6d0e4c23eef946ec32a8ae3585c5788c12e',
s: '0x43d5189f6e3c0fd5170c5d64ae0dc872d470a5799af679b547409354bdf1ad94',
v: 1,
creates: null,
chainId: 296,
wait: [Function (anonymous)]
}

```

However, there is nothing we really need to do with this information.

Let's query both MINFEE and totalBurnt again. The value returned for MINFEE is expected to be the same, since there are no functions which modify its value, and in fact it is marked as constant. The value returned for totalBurnt, on the other hand, is expected to increase from its previous value, since the balance of the smart contract should increase each time the burninate function is successfully invoked.

Query MINFEE.

```

javascript
await trogdor.functions.MINFEE();

```

This returns the same value of 100, as expected.

```

javascript
[ BigNumber { value: "100" } ]

```

Query totalBurnt.

```

javascript
await trogdor.functions.totalBurnt();

```

This returns a new value of 123, as expected as well.

```

javascript
[ BigNumber { value: "123" } ]

```

Let's also query the amounts function, which was auto-generated by the Solidity compiler for the mapping of the same name. This function expects an address parameter. Let's use the address of the same Hedera EVM account that we used to invoke the burninate function.

```

javascript
acc0EvmAddress = (await hre.ethers.getSigners())[0].address;
await trogdor.functions.amounts(acc0EvmAddress);

```

This returns a value of 123, which is expected since that was the value that was sent along with the transaction when invoking burninate

```

javascript
[ BigNumber { value: "123" } ]

```

Let's repeat the above, this time using the address of a different Hedera EVM account which has not invoked the burninate function yet.

```

javascript

```

```
acc1EvmAddress = (await hre.ethers.getSigners())[1].address;
await trogdor.functions.amounts(acc1EvmAddress);
```

This returns a value of 0, which is expected since this account has not yet invoked `burninate`.

```
javascript
[ BigNumber { value: "0" } ]
```

Using Hashscan to check smart contract interactions

Visit the "Contract" page for your previously deployed smart contract, e.g.

<https://hashscan.io/testnet/contract/0.0.15426051>

Scroll down to the "Recent Contract Calls" section

If you see "REFRESH PAUSED" at the top right of this section, press the "play" button next to it to unpause (otherwise it does not load new transactions)

You should see a list of transactions, with most recent at the top

There should be a successful transaction, denoted by the absence of an exclamation mark in a red triangle, e.g.

<https://hashscan.io/testnet/transaction/1689667816.410045835>

Scroll down to the "Contract Result" section

You should see "Result" as SUCCESS

You should also see "Error Message" as None

Scroll down to the "Logs" section

You should see a single log entry (address, data, index, and topics)

The "Address" field matches that of the smart contract

The "Index" field should be 0 since there was only a single event that was

emitted

The "Topics" field corresponds to the hash of the signature of the event that is emitted, e.g. `Burnination(address,uint256)`

The "Data" field corresponds to the values of the event parameters, e.g.

[illegible]

0x7394111093687e9710b7a7aeba3ba0f417c54474 (your address) and 0x7b is the amount (123 when converted to decimal)

```
# hederajs-sdk:
```

— — —

```
description: >-
```

Hedera SDK JS tutorial - HSCS workshop. Learn how to enable custom logic & processing on Hedera through smart contracts.

— — —

Hedera SDK JS

Video

```
{% embed url="https://www.youtube.com/watch?v=wRP7HyPjwu8" %}
Hedera Smart Contract Service Workshop Part 4/6 | Hedera SDK
{% endembed %}
```

Hedera SDK JS

The Hedera network offers multiple services:

Hedera Smart Contract Service (HSCS)

Hedera File Service (HFS)

Hedera Token Service (HTS)

Hedera Consensus Service (HCS)

Each service defines a number of different ways you can interact with it as a developer, and these comprise the Hedera Application Programming Interfaces (HAPIs). However, HAPIs are very close to the metal, and a developer needs to handle gRPCs and protocol buffers (among other things) to work with them successfully. Thankfully there are Hedera SDKs, which abstract these low-level complexities away. These SDKs allow you to interact with the various Hedera services via APIs exposed in a variety of different programming languages.

<details>

<summary>Available Hedera SDKs</summary>

At the time of writing, July 2023, these Hedera SDKs are available in the following languages:

- Hedera SDK JavaScript/ TypeScript
- Hedera SDK Java
- Hedera SDK Go
- Hedera SDK Swift

Please refer to SDKs for an up to date list of SDKs, including additional community-maintained SDKs.

</details>

In this tutorial, you will be using Hedera SDK JS to interact with HSCS. Specifically, you will use it to deploy a smart contract, query its state by invoking functions, and modify its state by invoking other functions.

Prerequisites

- ✓ Complete the Setup section of this same tutorial.
- ✓ Complete the Solidity section of this same tutorial.

Set up the project

To follow along, enter the `hederasdkjs` directory within the accompanying tutorial GitHub repository, which you should already have cloned in the Intro section earlier.

```
shell
cd ./hederasdkjs
npm install
```

Compiling smart contracts

Step D1: Copy smart contract

We have already written the smart contract in the Intro section of this tutorial. Let's copy that into this directory so that we may continue working on it.

```
shell
cp ../intro/trogdor.sol ./trogdor.sol
```

Using `solc` to compile smart contracts

Let's install the Solidity compiler, `solc` from npm.

```
shell
```

```
npm install --global solc@0.8.17
```

You can verify that it has installed successfully by asking it to output its version. Note that while the package name on npm is solc, the executable present on PATH is spelled slightly differently: solcjs.

```
shell
solcjs --version
```

If it does not error, and outputs its version, you know it has installed successfully.

0.8.17+commit.8df45f5f.Emscripten.clang

Let's explore its command line interface:

```
shell
solcjs --help
```

There are relatively few flags and options. In this tutorial, you will only be using --bin, and --abi.

Usage: solcjs [options]

Options:

-V, --version	output the version number
--version	Show version and exit.
--optimize	Enable bytecode optimizer. (default:
false)	
--optimize-runs <optimize-runs>	The number of runs specifies roughly how
often each opcode of the deployed code	will be executed across the lifetime of
the contract. Lower values will optimize	more for initial deployment cost, higher
values will optimize more for	high-frequency usage.
--bin	Binary of the contracts in hex.
--abi	ABI of the contracts.
--standard-json	Turn on Standard JSON Input / Output
mode.	
--base-path <path>	Root of the project source tree. The
import callback will attempt to interpret	all import paths as relative to this
directory.	
--include-path <path...>	Extra source directories available to the
import callback. When using a package	manager to install libraries, use this
option to specify directories where	packages are installed. Can be used
multiple times to provide multiple	locations.
-o, --output-dir <output-directory>	Output directory for the contracts.
-p, --pretty-json	Pretty-print all JSON output. (default:
false)	
-v, --verbose	More detailed console output. (default:
false)	
-h, --help	display help for command

This bytecode is used to deploy the smart contract onto the Hedera network.

<details>

<summary>EVM bytecode categories</summary>

The EVM bytecode that is output by the Solidity compiler is not the same as the EVM bytecode that is stored and executed on the network after it has been deployed.

The Solidity compiler's output bytecode is creation bytecode, sometimes also referred to as init bytecode.

The bytecode that is stored on the network is runtime bytecode, sometimes also referred to as deployed bytecode.

Ref: Stackoverflow: What is the difference between bytecode, init code, deployed bytecode, creation bytecode, and runtime bytecode?

</details>

Examine the ABI output

Open trogdorsolTrogdor.abi

If you are using a POSIX-compliant shell, and have jq installed, you can view the ABI output like so.

```
shell
jq < ./trogdorsolTrogdor.abi
```

The ABI essentially tells any user/ developer who wishes to interact with the EVM bytecode, what the exposed interface is. In fact ABI stands for Application Binary Interface. This interface will include any functions and events, which are needed by any clients (e.g. DApps), or other smart contracts, to be able to interact with it.

```
{% hint style="info" %}
Ref: Solidity - Contract ABI specification
{% endhint %}
```

```
json
[
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "address",
        "name": "who",
        "type": "address"
      },
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "amount",
        "type": "uint256"
      }
    ],
    "name": "Burnination",
    "type": "event"
  },
  {
```

```

    "inputs": [],
    "name": "MINFEE",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "name": "amounts",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "burninate",
    "outputs": [],
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "totalBurnt",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  }
]

```

This is extremely useful, because by examining the bytecode, which is what is deployed onto the Hedera network, you are likely to have no idea what it does, or how to interact with it. If you have the corresponding ABI in hand, however, you will have a very good idea of how you can interact with this smart contract, and perhaps can infer what it does as well.

Deploying smart contracts

Let's edit the `deploy-sc.js` file. In this script, you'll use Hedera SDK JS to deploy your smart contract onto Hedera Testnet.

Step E1: Initialise operator account

This script has already been set up to read in environment variables from the .env file that you have set up in the Intro section of this tutorial via the dotenv npm package, and they are now accessible using process.env.

We will use the OPERATORID and OPERATORKEY environment variables to initialise an operator account, connect to Hedera Testnet.

```
js
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromString(process.env.OPERATORKEY);
const client = Client.forTestnet();
client.setOperator(operatorId, operatorKey);
```

Step E2: Read the EVM bytecode from file

One of the outputs from running solc earlier was the binary file, which contains EVM bytecode. Let's read this from disk into memory.

```
js
const evmBytecode = await fs.readFile(
  './trogdorsolTrogdor.bin', { encoding: 'utf8' });
```

Step E3: Use HFS to store EVM bytecode on network

Next, write the EVM bytecode onto Hedera Testnet using HFS. In order to do so, you will need to use FileCreateTransaction.

```
{% hint style="info" %}
Note that you can use FileCreateTransaction for any type of file that is up to
1024KB in size. You are not restricted to only EVM bytecode.
{% endhint %}
```

```
js
const fileCreate = new FileCreateTransaction()
  .setContents(evmBytecode.toString());
const fileCreateTx = await fileCreate.execute(client);
const fileCreateReceipt = await fileCreateTx.getReceipt(client);
console.log('HFS FileCreateTransaction', fileCreateReceipt);
const fileId = fileCreateReceipt.fileId;
```

In the final line above, obtain the file ID from the FileCreateTransaction's receipt, as fileId - you will need it later.

Step E4: Deploy a smart contract on HSCS by referencing the bytecode on HFS

Now we're finally able to deploy the smart contract onto HSCS. In order to do so, you will need to use ContractCreateTransaction.

```
js
const scDeploy = new ContractCreateTransaction()
  .setBytecodeFileId(fileId)
  .setGas(100000);
const scDeployTx = await scDeploy.execute(client);
const scDeployReceipt = await scDeployTx.getReceipt(client);
console.log('HSCS ContractCreateTransaction', scDeployReceipt);
const scId = scDeployReceipt.contractId;
```

The fileId that you obtained in the previous step references the EVM bytecode stored on HFS. The ContractCreateTransaction references this file on HFS during the deployment process.

In the final line above, obtain the smart contract ID from the ContractCreateTransaction's receipt, as scId - you will need it later.

The smart contract is now deployed, and ready to be interacted with.

Run the script.

```
shell
node ./deploy-sc.js
```

You should see output similar to the following, which contains:

```
HFS FileCreateTransaction TransactionReceipt
HSCS ContractCreateTransaction TransactionReceipt
Deployed to
```

```
HFS FileCreateTransaction TransactionReceipt {
  status: Status { code: 22 },
  accountId: null,
  fileId: FileId {
    shard: Long { low: 0, high: 0, unsigned: false },
    realm: Long { low: 0, high: 0, unsigned: false },
    num: Long { low: 474925, high: 0, unsigned: false },
    checksum: null
  },
  contractId: null,
  topicId: null,
  tokenId: null,
  scheduleId: null,
  exchangeRate: ExchangeRate {
    hbars: 30000,
    cents: 169431,
    expirationTime: 2023-08-12T03:00:00.000Z,
    exchangeRateInCents: 5.6477
  },
  topicSequenceNumber: Long { low: 0, high: 0, unsigned: false },
  topicRunningHash: Uint8Array(0) [],
  totalSupply: Long { low: 0, high: 0, unsigned: false },
  scheduledTransactionId: null,
  serials: [],
  duplicates: [],
  children: []
}
HSCS ContractCreateTransaction TransactionReceipt {
  status: Status { code: 22 },
  accountId: null,
  fileId: null,
  contractId: ContractId {
    shard: Long { low: 0, high: 0, unsigned: false },
    realm: Long { low: 0, high: 0, unsigned: false },
    num: Long { low: 474926, high: 0, unsigned: false },
    evmAddress: null,
    checksum: null
  },
  topicId: null,
  tokenId: null,
  scheduleId: null,
  exchangeRate: ExchangeRate {
```

```

    hbars: 30000,
    cents: 169431,
    expirationTime: 2023-08-12T03:00:00.000Z,
    exchangeRateInCents: 5.6477
  },
  topicSequenceNumber: Long { low: 0, high: 0, unsigned: false },
  topicRunningHash: Uint8Array(0) [],
  totalSupply: Long { low: 0, high: 0, unsigned: false },
  scheduledTransactionId: null,
  serials: [],
  duplicates: [],
  children: []
}
Deployed to 0.0.474926

```

For both of the TransactionReceipt check that their status is { code: 22 }, which means that the transaction was successful.

The Deployed to outputs the account ID of the smart contract that you have just deployed.

Check smart contract deployment using Hashscan

Copy the output smart contract account ID, e.g. 0.0.15388539.
 Visit Hashscan
 Paste the copied account ID into the search box.
 You should get redirected to a "Contract" page, e.g.
<https://hashscan.io/testnet/contract/0.0.15388539>.
 In it you can see the EVM address, e.g.
 0x00eacf7b.
 Under "Contract Bytecode", you can see "Runtime Bytecode".

Interacting with smart contacts

Let's edit the interact-sc.js file. In this script, you'll use Hedera SDK JS to interact with your smart contract on Hedera Testnet.

Step F1: Specify deployed contract ID

Copy the smart contract ID, obtained during the previous step, and add paste this into this file, where the main function is invoked (at the bottom of the file).

```

js
  contractId: '0.0.15388539',

```

Step F2: Initialise operator account

Similar to what we did in the deployment script, we will use the OPERATORID and OPERATORKEY environment variables to initialise an operator account, connect to Hedera Testnet.

```

javascript
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorPrivateKey = PrivateKey.fromString(process.env.OPERATORKEY);
const client = Client.forTestnet();
client.setOperator(operatorId, operatorPrivateKey);

```

Step F3: Invoke payable function with zero value

The burninate function in this smart contract is public and payable. This means

that the function may be invoked with a transaction that has a value (HBAR) attached to it - accessible as `msg.value` in Solidity. The value will be added to this smart contracts balance if this function is executed successfully.

Recall that when you implemented the `burninate` function in the Intro section of this tutorial, that there is this `require` statement:

```
solidity
require(msg.value >= MINFEE, "pay at least minimum fee");
```

This essentially specifies that the function will error, and therefore not execute successfully, when the value sent with the transaction is anything less than 100 tinybar (`MINFEE`).

Now we're going to invoke this function with a zero value transaction, i.e. Invoke `burninate` with `msg.value = 0`. This is done on purpose, to trip up this `require` statement, so that we can witness the rejection.

To do so, use `ContractExecuteTransaction`.

```
javascript
const scWrite1 = new ContractExecuteTransaction()
  .setContractId(contractId)
  .setGas(100000)
  .setFunction(
    'burninate',
    new ContractFunctionParameters(),
  );
const scWrite1Tx = await scWrite1.execute(client);
```

This will send a transaction to Hedera Testnet, which contains a request to HSCS to (potentially) modify the state of this smart contract.

When this is run, we expect the transaction to fail, with a `CONTRACTREVERTEXECUTED` error. The reason for this is the `require` statement in this function, as described above - we need to send some HBAR!

Step F4: Invoke payable function with non-zero value

Next invoke the same `burninate` function once again, with only one change: the transaction will contain a value of 123 tinybars, i.e. `msg.value = 123`.

This time, the function invocation will succeed, as it passes that `require` statement.

```
js
const scWrite2 = new ContractExecuteTransaction()
  .setContractId(contractId)
  .setGas(100000)
  .setPayableAmount(new Hbar(123, HbarUnit.Tinybar))
  .setFunction(
    'burninate',
    new ContractFunctionParameters(),
  );
const scWrite2Tx = await scWrite2.execute(client);
```

Step F5: Invoke view function with no parameters

The `burninate` function is one that can (and does) modify the persisted state of the smart contract. However there are other functions which do not do so, and instead merely read (query) the currently persisted state of the smart contract.

These functions have the view modifier.

```
{% hint style="info" %}
```

There are also other functions which neither read the currently nor modify the persisted state of the smart contract. These functions have the pure modifier.

These are typically used as utility functions, intended to be invoked by other functions within a smart contract.

```
{% endhint %}
```

The totalBurnt is a view function, and to invoke that, let's use ContractCallQuery.

```
{% hint style="info" %}
```

ContractExecuteTransaction: Use for modifying state

ContractCallQuery: Use for reading state

```
{% endhint %}
```

```
js
```

```
const scRead1 = new ContractCallQuery()  
  .setContractId(contractId)  
  .setGas(100000)  
  .setFunction(  
    'totalBurnt',  
    new ContractFunctionParameters(),  
  )  
  .setQueryPayment(new Hbar(2));  
const scRead1Tx = await scRead1.execute(client);  
const scRead1ReturnValue = scRead1Tx.getUint256();
```

Once the ContractCallQuery is executed, extract the its return value using the getter function with the appropriate type. Since the totalBurnt function specifies returns(uint256) in its signature, use getUint256() to extract that return value.

```
{% hint style="info" %}
```

The ContractCallQuery has setQueryPayment, which is to pay for the costs of querying the data. Note that this is different from other EVM-compatible networks, which allow you to query smart contract state without paying any fee.\

Ref: Hedera - Get the cost of requesting the query

```
{% endhint %}
```

Step F6: Convert account ID to EVM address

In the subsequent step, we will use the operator account as an input parameter in a function invocation. However, we need to convert this from an Account ID format, which looks like 0.0.3996280, to an EVM address format, which looks like 0x7394111093687e9710b7a7aeba3ba0f417c54474. This is because the EVM (and by extension Solidity), does not understand Hedera-native accounts. Instead it only understands EVM accounts.

To do so, we start with the private key of the operator account, from that we derive its public key, and finally from that we derive its EVM account. Thankfully Hedera SDK JS has utility functions for these, and the conversion can be performed quite easily.

```
javascript
```

```
const operatorPublicKey = operatorPrivateKey.publicKey;  
const operatorEvmAddress = operatorPublicKey.toEvmAddress();
```

Step F7: Invoke auto-generated view function with parameters

In this smart contract `amounts` is a view function, and to invoke that, let's use `ContractCallQuery`. There are a couple of key differences though:

The `amounts` function requires an input parameter, or type `address`

The `amounts` function was not written using Solidity code, But instead was auto-generated by the Solidity compiler for the public state variable with the same name.

<details>

<summary>Auto-generated getter function</summary>

This is the actual code for `amounts` in the Solidity file:

```
solidity
    mapping(address => uint256) public amounts;
```

This is what the auto-generated function for `amounts` would have looked like, if you needed to write it manually.

```
solidity
    function amounts(address account)
        public
        view
        returns(uint256) {
        // implementation goes here
    }
```

</details>

Let's send a `ContractCallQuery` to the `amounts` function. Use the `operatorEvmAddress` obtained in the previous step as the input parameter.

There is a `ContractFunctionParameters`, which we've used in the previous smart contract invocations, but it was always "empty", in the sense that there were no parameters. Since `amounts` requires a single parameter of type `address`, use `addAddress()` to specify its value.

```
js
const scRead2 = new ContractCallQuery()
    .setContractId(contractId)
    .setGas(100000)
    .setFunction(
        'amounts',
        new ContractFunctionParameters()
            .addAddress(operatorEvmAddress),
    )
    .setQueryPayment(new Hbar(2));
const scRead2Tx = await scRead2.execute(client);
const scRead2ReturnValue = scRead2Tx.getUint256();
```

Once the `ContractCallQuery` is executed, extract the its return value using the getter function with the appropriate type. The `amounts` mapping specifies `uint256` as its value type, this is equivalent to a function specifying `returns(uint256)` in its signature. Use `getUint256()` to extract that return value.

Run the script.

shell

```
node ./interact-sc.js
```

You should get output similar to the following:

```
ContractExecuteTransaction #1 ReceiptStatusError
ContractExecuteTransaction #2 TransactionReceipt
ContractCallQuery #1 ContractFunctionResult
return value
```

```
ContractExecuteTransaction #1 ReceiptStatusError: receipt for transaction
0.0.1186@1691806933.486622108 contained error status CONTRACTREVERTEXECUTED
  at new ReceiptStatusError (/Users/user/code/hedera/hedera-smart-contracts-
workshop/hederasdkjs/node_modules/@hashgraph/sdk/lib/ReceiptStatusError.cjs:43:5)
  at TransactionReceiptQuery.mapStatusError (/Users/user/code/hedera/hedera-
smart-contracts-workshop/hederasdkjs/node_modules/@hashgraph/sdk/lib/
transaction/TransactionReceiptQuery.cjs:276:12)
  at TransactionReceiptQuery.execute (/Users/user/code/hedera/hedera-smart-
contracts-workshop/hederasdkjs/node_modules/@hashgraph/sdk/lib/
Executable.cjs:671:22)
  at process.processTicksAndRejections (node:internal/process/taskqueues:95:5)
  at async TransactionResponse.getReceipt (/Users/user/code/hedera/hedera-
smart-contracts-workshop/hederasdkjs/node_modules/@hashgraph/sdk/lib/
transaction/TransactionResponse.cjs:86:21)
  at async main
(/Users/user/code/hedera/hedera-smart-contracts-workshop/hederasdkjs/interact-
sc.js:48:29) {
  status: Status { code: 33 },
  transactionId: TransactionId {
    accountId: AccountId {
      shard: [Long],
      realm: [Long],
      num: [Long],
      aliasKey: null,
      evmAddress: null,
      checksum: null
    },
    validStart: Timestamp { seconds: [Long], nanos: [Long] },
    scheduled: false,
    nonce: null
  },
  transactionReceipt: TransactionReceipt {
    status: Status { code: 33 },
    accountId: null,
    fileId: null,
    contractId: ContractId {
      shard: [Long],
      realm: [Long],
      num: [Long],
      evmAddress: null,
      checksum: null
    },
    topicId: null,
    tokenId: null,
    scheduleId: null,
    exchangeRate: ExchangeRate {
      hbars: 30000,
      cents: 169431,
      expirationTime: 2023-08-12T03:00:00.000Z,
      exchangeRateInCents: 5.6477
    },
    topicSequenceNumber: Long { low: 0, high: 0, unsigned: false },
    topicRunningHash: Uint8Array(0) [],
```

[illegible]

hashscan.io).

outro.md:

description: >-

Outro - HSCS workshop. Learn how to enable custom logic & processing on Hedera through smart contracts.

Outro

Video

{% embed url="https://www.youtube.com/watch?v=s1xkAl9IEc" %}
Hedera Smart Contract Service Workshop Part 6/6 | Conclusion
{% endembed %}

Congrats!

You've now completed this workshop, and you're now able to:

- Write the code for a smart contract using solidity.
- Compile the solidity code into EVM bytecode.
- Deploy the smart contract onto HSCS.
- Interact with the smart contract on HSCS to query and change its state.

If you've completed this entire tutorial, wondering what Trogdor was all about...

- Watch the Trogdor video
- Play the Trogdor game

Where to go from here

There's much more involved in developing smart contracts - here are several things to consider looking into after completing this workshop.

Solidity is a fairly compact language in terms of syntax, and we've covered the essentials already. However there is still quite a bit that was not covered here:

- constructor
- function parameters
- structs
- conditional logic
- iterative logic
- custom function modifiers
- interface
- import
- inheritance
- smart contracts interacting with other smart contracts
- ... and more

The Solidity language documentation is a great reference to explore on this front

Since a smart contract's code is immutable once deployed, it is extremely important to "get it right" prior to deployment. Therefore testing smart contracts becomes even more important. Hardhat has a built-in testing facility (test runner plus various test utilities and conveniences) that we have not explored here, and has a great smart contract testing tutorial.

Owing to the same immutable aspect mentioned above, smart contract security is also very important. Look into security audits, static analysis, and dynamic analysis methods.

Finally, verifying smart contracts such that their code and ABI are publicly available on the network explorer is important in building trust in the ecosystem. However, this feature is still a work in progress on Hashscan at the time of writing (July 2023). When this feature lands, we'll have other niceties as well, for example event logs displayed on transactions can show their parsed values, instead of their raw form.

README.md:

description: >-

Hedera Smart Contract Service (HSCS) workshop. Learn how to enable custom logic & processing on Hedera through smart contracts.

Hedera Smart Contracts Workshop

Smart contracts are a means to enable custom logic and processing in a DLT. Developers can harness their power to build their own decentralized applications (DApps). Learn how to get started with the Hedera Smart Contract Service (HSCS) in this workshop.

What we will cover

- Syntax of Solidity, a programming language used to write smart contracts
- Using the Solidity compiler
- Using Hedera SDK JS to deploy and interact with smart contracts on Hedera networks
- Using Hardhat + EthersJS to deploy and interact with smart contracts on Hedera networks
- Where to go from here

Video

```
{% embed url="https://www.youtube.com/watch?v=2nFx6aJdx9U" %}  
Hedera Smart Contract Service Workshop Part 1/6 | Introduction  
{% embed %}
```

Prerequisites

Prior knowledge:

- ✓ Javascript syntax
- ✓ Hedera network core concepts

System setup:

- ✓ git installed
 - Minimum version: 2.37
 - Install Git (Github)
- ✓ NodeJs + npm installed
 - Minimum version of NodeJs: 18
 - Minimum version of npm: 9.5
 - Recommended for Linux & Mac: nvm
 - Recommended for Windows: nvm-windows
- ✓ POSIX-compliant shell
 - For Linux & Mac: The shell that ships with the operating system will work. Either bash or zsh will work.
 - For Windows: The shell that ships with the operating system (cmd.exe,

powershell.exe) will not work. Recommended alternatives: WSL/2, or git-bash which ships with git-for-windows.

- ✓ Internet connection

- ✓ Optionally, jq

For Linux: Use OS package manager

For Mac: brew install jq

For Windows: Install .exe file manually: JQ releases (Github)

Software libraries and developer tools

Before we begin coding, let's take a look at the various software libraries and developer tools that you will need to be familiar with when working with smart contracts on HSCS.

Hedera SDK JS is a software library that contains functions designed to interact the all of the services available on the Hedera network: HCS, HTS, HFS, and HSCS. That includes smart contracts.

Both EthersJs and Web3Js are software libraries that contain functions designed to interact with the Ethereum network, and any other EVM-compatible networks. This means that you can use them to interact with HSCS as well (but not with HCS, HTS, or HFS).

Smart contracts are written using Solidity, but we cannot just take the Solidity code and ask HSCS, or any other EVM implementation, to run it. Instead we need solc, the Solidity compiler, to compile it into EVM bytecode, which can then be executed by any EVM implementation, including the one in HSCS.

<details>

<summary>Other smart contract programming languages</summary>

Solidity is not the only game in town. You can actually write smart contracts in any language, as long as it can compile to EVM bytecode. The most popular alternative smart contract programming language is Vyper.

</details>

In this workshop we will be using Solidity.

Truffle Suite, Hardhat, and Foundry are developer frameworks that are designed to make it easier to work with smart contract development workflows by providing various utilities, scripts, structure, and documentation that are useful during development.

In this workshop we will be using Hardhat.

<p>Writer: Brendan, DevRel Engineer</p> <p>GitHub Blog</p>	<p>Editor: Michiel, Developer Advocate</p> <p>GitHub LinkedIn</p>

setup.md:

description: >-
Setup tutorial - HSCS workshop. Learn how to enable custom logic & processing
on Hedera through smart contracts.

Setup

 Video

```
{% embed url="https://www.youtube.com/watch?v=25y5zsl0Uj8" %}  
Hedera Smart Contract Service Workshop Part 2/6 | Setup  
{% endembed %}
```

Set up the project

The setup has already been (mostly) done. All that's left for you to do is clone the accompanying tutorial GitHub repository and install the dependencies:

```
sh  
git clone -b main git@github.com:hedera-dev/hedera-smart-contracts-workshop.git  
cd hedera-smart-contracts-workshop
```

```
{% hint style="info" %}  
If you do not have SSH available on your system, or are unable to configure it  
for GitHub, you may wish to try this git command instead:
```

```
git clone -b main https://github.com/hedera-dev/hedera-smart-contracts-  
workshop.git  
{% endhint %}
```

Configuration

Step B1: Environment variables file

In the root directory of the repo, you will find a file named `.env.example`. Make a copy of this file, and name it `.env`.

```
shell  
cp .env.example .env
```

Step B2: Operator account

An operator account is used to obtain an initial sum of HBAR on Hedera Testnet, and then use that to pay for various Hedera network operations. This includes everything from basic transactions, to gas fees for HSCS interactions.

Visit the Hedera Portal to get started.

(1) Create a Testnet account.

<figure><figcaption><p>Hedera Portal - Create Testnet Account</p></figcaption></figure>

(2) Copy-paste the confirmation code sent to your email.

<figure><figcaption><p>Hedera Portal - Email Verification</p></figcaption></figure>

(3) Fill out this form with details for your profile.

<figure><figcaption><p>Hedera Portal - Profile Details</p></figcaption></figure>

(4) In the top-left there is a drop down menu, select between Hedera Testnet (default) and Previewnet:

<figure><figcaption><p>Hedera Portal - Select Network</p></figcaption></figure>

(5) From the next screen that shows your accounts, copy the value of the "DER-encoded private key" and replace OPERATORKEY in the .env file with it.

<figure><figcaption><p>Hedera Portal - Account Details</p></figcaption></figure>

(6) From the same screen, copy the value of "Account ID" and replace the value of the OPERATORID variable in the .env file with it.

{% hint style="info" %}

Note that private keys should be stored and managed securely. For the purposes of a tutorial, secure key management has been skipped, and you are storing your private keys in plain text on disk. Do not do this in production applications.

{% endhint %}

Step B3: Seed phrase

When developing smart contracts, you often need more than 1 account to do so. Thankfully we do not need to go through the somewhat cumbersome process of creating multiple accounts via the Hedera Portal - you only really need to do that once for the operator account.

Any subsequent accounts that you wish to create can be generated programmatically, and funded with HBAR from your operator account.

To do so, we will utilise something called a seed phrase, which is a sequence of selected dictionary words chosen at random. This process is defined in BIP-39.

Subsequently, we will use that seed phrase as an input and generate multiple accounts; each of which consists of a private key, a public key, and an address. This process is defined in BIP-44.

<details>

<summary>BIP, EIP, and HIP</summary>

"BIP" stands for Bitcoin Improvement Proposal.

"EIP" stands for Ethereum Improvement Proposal, and was preceded by "ERC" which stands for Ethereum Request for Comments.

"HIP" stands for Hedera Improvement Proposal.

</details>

Interestingly these 2 BIPs were never adopted by the Bitcoin community, but are almost de-facto used by everyone in the Ethereum community. On Hedera, you can use these 2 BIPs to generate Hedera EVM accounts, but this is not possible for Hedera-native accounts (as they use a different type of public key algorithm).

<details>

<summary>ECDSA and EdDSA</summary>

ECDSA (Elliptic Curve Digital Signing Algorithm) is a public key algorithm, and secp256k1 is a particular configuration that may be used by the ECDSA algorithm.

EdDSA (Edwards Digital Signing Algorithm) is another public key algorithm, and Ed25519 is a particular configuration that may be used by the EdDSA algorithm.

Both Bitcoin and Ethereum use ECDSA with secp256k1 for their accounts.

Hedera native accounts use EdDSA with Ed25519, and Hedera EVM accounts use ECDSA with secp256k1.

</details>

Enough theory - let's generate a seed phrase!

Visit iancoleman.io/bip39, and you can generate a BIP39 seed phrase there: \

<figure><figcaption></figcaption></figure>

\

Locate the line that is labelled "Generate a random mnemonic"
Select any number from the dropdown that is more than or equal to 12
Press "GENERATE"
Locate the section that is labelled "BIP39 Mnemonic"
Copy these words from the text box - this will be your BIP39 seed phrase

Replace the value of the BIP39SEEDPHRASE variable in the .env file with this phrase.

```
{% hint style="info" %}  
Ref: BIP-39 Mnemonic code for generating deterministic keys  
Ref: BIP-44: Multi-Account Hierarchy for Deterministic Wallets  
{% endhint %}
```

Step B4: Fund several Hedera EVM accounts

At this point, you have an operator account, which is already funded with HBAR, and you have a seed phrase. Let's generate more accounts based on the seed phrase, and then transfer HBAR to them from the operator account.

First switch to the intro directory, and install dependencies using npm.

```
cd ./intro  
npm install
```

Next, let's use a script already prepared for you. We want this script to generate 2 Hedera EVM accounts, and transfer 100 HBAR to each of them, so let's set those values in generate-evm-accounts.js.

```
javascript  
const NUMACCOUNTS = 2;  
const AMOUNTPERACCOUNT = 100;  
const HDPATH = "m/44'/60'/0'/0";
```

<details>

<summary>Choice of derivation path</summary>

Above, we're using `m/44'/60'/0'/0` as the derivation path. This value is the Ethereum derivation path, and we need to use this because Metamask does not allow it to be configured.

</details>

Run this script.

```
shell
node ./generate-evm-accounts.js
```

This should output something similar to the following:

```
EVM account #0 generated.
#0      HD path: m/44'/60'/0'/0/0
#0 Private key:
3030020100300706052b8104000a04220420fb11afc5d508036ac7a9df9f1eb7cea551e4a7b738c2
c70da099fe5f379f3364
#0 Public key:
302d300706052b8104000a032200027a753c29cc9f0ea0b6ccf0614676daeba3da0dbd5f54ef9850
ad3878ded4e077
#0 EVM address: 07ffaadfe3a598b91ee08c88e5924be3eff35796
EVM account #1 generated.
#1      HD path: m/44'/60'/0'/0/1
#1 Private key:
3030020100300706052b8104000a042204206e3ff9f1f1ae58248a5838ec877acc55d10300958622
4d76ab74a652d408cf12
#1 Public key:
302d300706052b8104000a03220002c4c2ed7a682a601c9c61dec42e87442b63893a6e5efdf6dc32
7a4b3bcc62aba9
#1 EVM address: 1c29e31d241f0d06f3763221f5224a6b82f09cce
Transfer transaction ID: 0.0.3996280@1690161480.080071857
HashScan URL:
https://hashscan.io/testnet/transaction/0.0.3996280@1690161480.080071857
```

Check funding of EVM accounts on Hashscan

Copy the HashScan URL, paste it into a browser, and you will see a "Transaction" page on HashScan.

<figure><figcaption><p>Screenshot showing a single transaction with multiple recipients transferring HBAR (on hashscan.io).</p></figcaption></figure>

Scroll down to the "Transfers" section, which should show the flow of HBAR between various accounts. In this case -200 (and a fractional amount of -0.00185217) from the operator account, +100.00000000 to each of the 2 EVM accounts, and fractional amounts to a couple of other accounts to pay for transaction processing. (Note that the fractional amounts may vary, they won't necessarily be 0.00185217 as above.)

Now you should have 1 Hedera-native account (previously funded), plus 2 new EVM accounts (freshly funded).

Address formats

Hedera networks have a native account address format, called the Account ID. An example of this would be: 0.0.3996280.

They are, at their core, simply computer programs that are executable. So what is the big deal about them then? What makes them different from "regular" computer programs? The biggest one, that you probably know already, is that they are executed on blockchains/ DLTs. But there are a few others to be aware of:

- They are typically executed within a Virtual Machine (VM)
- Any state changes need to be agreed upon through network consensus
- Any state queries return values agreed upon by network consensus
- While their state is mutable, their code is not

Combine the above with the decentralized nature of blockchains/ DLTs, and you get a special breed of computer programs like no other: Deterministic, p2p execution, that is censorship resistant and interruption resistant.

You can use this powerful technology within the Hedera network too, via the Hedera Smart Contract Service. This workshop will show you how!

To start, open and edit `intro/trogdor.sol`.

Comments

In Solidity, comment syntax is similar to what you might be familiar with from Javascript. Single line comments use `//`, and extend till the rest of the line. Multi-line comments use `/` to begin, and `/` to end.

```
solidity
// single line comment

/ this is a
multi-line
comment. /
```

SPDX License

SPDX defines a list of software licenses, and allows you to reference one using a standard short identifier. The solidity compiler will explicitly check for this as a comment in the first line of any Solidity file. If it is missing, it will output a warning.

```
solidity
// SPDX-License-Identifier: MIT
```

Step A1: Specify solc version number

```
{% hint style="info" %}
Near the top of the file, you should see the following comment:
```

```
solidity
// NOTE: specify solc version number
// Step (A1) in the accompanying tutorial
```

Note that this type of comment will be present throughout the tutorial repo that accompanies this written tutorial. Each numbered step of a section heading here corresponds to the the same number in a comment there. In the subsequent steps of this tutorial, you will follow the same pattern as above. However, this tutorial does not repeat the comments marking the steps for the remainder of the tutorial and instead only include the new/changed lines of code.

```
{% endhint %}
```

The pragmas simply defines the which version of the Solidity compiler, solc, it is intended to be compiled with.

```
solidity
pragma solidity 0.8.17;
```

Here we simply specify that this file should be compiled with version 0.8.17 only. You may specify more complex rules, similar to semver used by npm.

```
{% hint style="info" %}
  Ref: Solidity version pragma
{% endhint %}
```

Step A2: Specify name of smart contract

A smart contract is a grouping of state variables, functions, and other things. The solidity code needs to group them, and does so by surrounding them with a pair of squiggly brackets ({ and }). It also needs a name - and we'll name this one Trogdor.

```
solidity
contract Trogdor {
```

Step A3: Primitive type state variable

A smart contract persists its state on the virtual machine, and may only be modified during a successful transaction on the network. Solidity supports many different primitive types, here let's use uint256 as we'll be representing an unsigned integer value.

```
solidity
    uint256 public MINFEE = 100;
```

The above would work, but in this case, we know that we will not be modifying its value, so instead of a state variable, let's use a state constant instead. This is achieved by adding the constant keyword to it.

```
solidity
    uint256 public constant MINFEE = 100;
```

```
{% hint style="info" %}
  Ref: Solidity value types
{% endhint %}
```

Step A4: Dynamic type state variable

Smart contracts can also persist more complex types of data in its state, and this is accomplished using dynamic state variables. A mapping is used to represent key-value pairs, and is analogous to a Hashmap in other programming languages.

```
solidity
    mapping(address => uint256) public amounts;
```

This mapping stores key-value pairs where the keys are of type address, and the values are of type uint256.

Note that both MINFEE and amounts have a visibility modifier of public. In other cases you might want internal or private,

```
{% hint style="info" %}
```

```
Ref: Solidity mapping type
Ref: Solidity State Variable Visibility
{% endhint %}
```

Functions

Functions are the main part of the smart contract where things actually happen: Code is executed, and perhaps state is accessed or updated. The syntax of a function is somewhat similar to Javascript, with the main differences being the addition of types, and of modifiers.

```
solidity
function doSomething(uint256 param1)
    public
    pure
    returns(uint256)
{
    return param1;
}
```

In the above example:

```
Name: doSomething
Modifiers: public and pure
Parameter name: param1
Parameter type: uint256
Return type: uint256
```

Step A5: Specify function modifiers

The burninate function modifies the state of the smart contract, and also accepts payment (in HBAR), so let's go with public, payable for its modifiers.

```
solidity
    public
    payable
```

The totalBurnt function does not modify the state of the smart contract, but does access the state. It does not accept any payment. It is intended to only be called by an Externally Owned Account (EOA) or another smart contract. For this let's go with external, view for its modifiers.

```
solidity
    external
    view
```

Just like state variables, functions may have the have visibility modifiers public, private, and internal; however external is a new one, and may apply only to functions.

```
{% hint style="info" %}
Ref: Solidity function visibility modifiers
{% endhint %}
```

Step A6: Specify function return values

The totalBurnt function performs a query of the smart contract's state. Therefore it should reply with this information. This is done through the returns keyword, which specifies the type of the returned value.

```
solidity
```

```
returns(uint256)
```

In this case, the function returns a single value of type uint256. which specifies one or more return values. Note that Solidity allows functions to return multiple return values, for example `returns(uint256, address)` would mean that it returns both a uint256 and an address.

Special values accessible within a function

When a smart contract function is invoked, it has access to values that are passed in as parameters. It also has access to the state variables persisted by the smart contract.

Additionally, there are also several special values that are specific to the current block (group of transactions) or specific to the current transaction that are also accessible. Two of these are `msg.sender` and `msg.value`.

```
{% hint style="info" %}  
  Ref: Solidity block and transaction properties  
{% endhint %}
```

Step A7: Specify condition for require

The `require` function is used to check for specific conditions within a function invocation. If these conditions are not met, it throws an exception. This causes the function invocation to be reverted, meaning the state of the smart contract would remain as it was before, as if the function invocation was never made.

```
solidity  
    require(msg.sender != address(0), "zero address not allowed");
```

Within the `burninate` function, we use a `require` to ensure that the transaction is seemingly not from the null address, also known as the zero address. This essentially disallows any transactions sent from that particular address

```
{% hint style="info" %}  
Technically it should not be possible for a transaction to be sent by the zero  
address. This is done here purely for illustrative purposes.  
{% endhint %}
```

```
{% hint style="info" %}  
  Ref: Solidity panic and require  
{% endhint %}
```

Step A8: Specify error message for require

We have another `require` in this function to ensure that the amount paid (in HBAR) is at least above a certain threshold (the `MINFEE` constant).

```
solidity  
    require(msg.value >= MINFEE, "pay at least minimum fee");
```

This function is payable, meaning that any value (of HBAR) sent along with the function gets deducted from the balance of the sender's account, and gets added to the balance of the smart contract's account. In other words: The transaction sender pays into the smart contract via this function.

```
{% hint style="info" %}  
The numeric value of msg.value is not denominated in HBAR, but rather tinybar,  
when a smart contract is deployed on a Hedera network. This is consistent with  
msg.value being denominated not in Ether, but rather in wei, when a smart
```

contract is deployed on an Ethereum network.

There is a key difference though:

```
1 HBAR = 10^8 tinybar (10 million)
1 Ether = 10^18 wei (1 billion billion)
{% endhint %}
```

In functions which are not payable, `msg.value` is guaranteed to be zero. Whereas in functions which are payable, `msg.value` could be zero or more. In this case, the intent is for the function to reject any function invocations which do not pay enough.

```
{% hint style="info" %}
Ref: Solidity Ether units
Ref: Stackoverflow full unit of HBAR
{% endhint %}
```

Step A9: Update state

After the checks have been completed successfully, by the require statements, we're ready to update the persisted state of this smart contract. In this case, we are keeping track, as a running tally, of the total amount paid by each different address that this function has been invoked with.

```
solidity
    amounts[msg.sender] = amounts[msg.sender] + msg.value;
```

This statement increments the current value by the amount paid into the function, keyed on the address that invoked this function.

```
{% hint style="info" %}
Ref: Solidity operators
Ref: Solidity mapping types
{% endhint %}
```

Step A10: Specify an event

The EVM outputs logs, which essentially is information that is persisted on the network, but not accessible by smart contracts. Instead they are intended to be accessed by client applications (such as DApps), which typically search for specific events, or listen for specific events.

The canonical use case for events within a smart contract is to create a "history" of actions performed by that smart contract. In this case, let's commemorate each time the burninate function is successfully invoked.

```
solidity
    event Burnination(address who, uint256 amount);
```

This event is named Burnination, and whenever it is produced, it is added to the EVM logs, and will contain an address and a uint256.

Step A11: Emit an event

Once the event has been defined, the smart contract should specify exactly when it should be added to the EVM logs. This is done using `emit`.

```
solidity
    emit Burnination(msg.sender, msg.value);
```

Thus, based on where this emit statement is located within the function, this event is added to the logs upon each time the burninate function is invoked, only if both of the require statements are satisfied. When it gets added the transaction sender's address and the amount that they paid into the function are logged.

```
{% hint style="info" %}
```

Note that this particular smart contract does not include any means to take out the HBAR balance that accrues within it over time each time burninate is invoked. This effectively means that the HBAR sent into it is stuck there forever, and hence is effectively lost.

```
Trogdor would be proud ;)  
{% endhint %}
```

create-and-transfer-an-nft-using-a-solidity-contract.md:

Create and Transfer an NFT using a Solidity Contract

Summary

Besides creating NFTs using Hedera SDK, you can use a Solidity Contract to create, mint, and transfer NFTs by calling contract functions directly. These are the contracts you will need to import into your working directory provided by Hedera that you can find in the contracts folder here:

```
HederaTokenService.sol  
HederaResponseCodes.sol  
IHederaTokenService.sol  
ExpiryHelper.sol  
FeeHelper.sol  
KeyHelper.sol
```

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

Get a Hedera testnet account.
Set up your environment here.

If you are interested in creating, minting, and transferring NFTs using Hedera SDKs you can find the example here.

```
{% hint style="warning" %}
```

In this example, you will set gas for smart contract transactions multiple times. If you don't have enough gas you will receive an<mark style="color:blue;">INSUFFICIENTGAS</mark> response. If you set the value too high you will be refunded a maximum of 20% of the amount that was set for the transaction.

```
{% endhint %}
```

1. Create an "NFT Creator" Smart Contract

You can find an NFTCreator Solidity contract sample below with the contract bytecode obtained by compiling the solidity contract using Remix IDE. If you are not familiar with Solidity, you can take a look at the docs here.

The following contract is composed of three functions:

```
<mark style="color:blue;">createNft</mark>
<mark style="color:blue;">mintNft</mark>
<mark style="color:blue;">transferNft</mark>
```

The important thing to know is that the NFT created in this example will have the contract itself as Treasury Account, Supply Key, and Auto-renew account. There's NO admin key for the NFT or the contract.

```
{% tabs %}
{% tab title="NFTCreator.sol" %}
solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.9.0;

import "./HederaResponseCodes.sol";
import "./IHederaTokenService.sol";
import "./HederaTokenService.sol";
import "./ExpiryHelper.sol";
import "./KeyHelper.sol";

contract NFTCreator is ExpiryHelper, KeyHelper, HederaTokenService {

    function createNft(
        string memory name,
        string memory symbol,
        string memory memo,
        int64 maxSupply,
        int64 autoRenewPeriod
    ) external payable returns (address){

        IHederaTokenService.TokenKey[] memory keys = new
        IHederaTokenService.TokenKey[](1);
        // Set this contract as supply for the token
        keys[0] = getSingleKey(KeyType.SUPPLY, KeyValueTypes.CONTRACTID,
        address(this));

        IHederaTokenService.HederaToken memory token;
        token.name = name;
        token.symbol = symbol;
        token.memo = memo;
        token.treasury = address(this);
        token.tokenSupplyType = true; // set supply to FINITE
        token.maxSupply = maxSupply;
        token.tokenKeys = keys;
        token.freezeDefault = false;
        token.expiry = createAutoRenewExpiry(address(this), autoRenewPeriod); //
        Contract auto-renews the token

        (int responseCode, address createdToken) =
        HederaTokenService.createNonFungibleToken(token);

        if(responseCode != HederaResponseCodes.SUCCESS){
            revert("Failed to create non-fungible token");
        }
        return createdToken;
    }

    function mintNft(
        address token,
        bytes[] memory metadata
    ) external returns(int64){
```



```
(int response, , int64[] memory serial) =
HederaTokenService.mintToken(token, 0, metadata);

    if(response != HederaResponseCodes.SUCCESS){
        revert("Failed to mint non-fungible token");
    }

    return serial[0];
}

function transferNft(
    address token,
    address receiver,
    int64 serial
) external returns(int){

    int response = HederaTokenService.transferNFT(token, address(this),
receiver, serial);

    if(response != HederaResponseCodes.SUCCESS){
        revert("Failed to transfer non-fungible token");
    }

    return response;
}
}

[% endtab %]

[% tab title="NFTCreatorsolNFTCreator.bin" %]

60806040523480156200001157600080fd5b5060018060008060068111156200002d576200002c62
0001f1565b5b6006811115620000425762000041620001f1565b5b81526020019081526020016000
208190555060026001600060016006811115620000715762000070620001f1565b5b600681111562
0000865762000085620001f1565b5b81526020019081526020016000208190555060046001600060
026006811115620000b557620000b4620001f1565b5b6006811115620000ca57620000c9620001f1
565b5b81526020019081526020016000208190555060086001600060036006811115620000f95762
0000f8620001f1565b5b60068111156200010e576200010d620001f1565b5b815260200190815260
200160002081905550601060016000600460068111156200013d576200013c620001f1565b5b6006
811115620001525762000151620001f1565b5b815260200190815260200160002081905550602060
01600060056006811115620001815762000180620001f1565b5b6006811115620001965762000195
620001f1565b5b815260200190815260200160002081905550604060016000600680811115620001
c457620001c3620001f1565b5b6006811115620001d957620001d8620001f1565b5b815260200190
81526020016000208190555062000220565b7f4e487b7100000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
60806040526004361061007b5760003560e01c806382b562aa1161004e57806382b562aa14610160
5780639b23d3d91461019d578063b6c907a6146101da578063eac6f3fe1461020a5761007b565b80
630a284cb61461008057806311e1fc07146100bd57806315dacbea146100fa578063618dc65e1461
0137575b600080fd5b34801561008c57600080fd5b506100a760048036038101906100a291906114
f9565b610247565b6040516100b49190611571565b60405180910390f35b3480156100c957600080
fd5b506100e460048036038101906100df91906115c2565b6102c9565b6040516100f19190611571
565b60405180910390f35b34801561010657600080fd5b50610121600480360381019061011c9190
6115c2565b6103e5565b60405161012e9190611571565b60405180910390f35b3480156101435760
0080fd5b5061015e60048036038101906101599190611629565b610503565b005b34801561016c57
600080fd5b50610187600480360381019061018291906116b1565b61062a565b6040516101949190
61171d565b60405180910390f35b3480156101a957600080fd5b506101c460048036038101906101
bf91906115c2565b610698565b6040516101d19190611571565b60405180910390f35b6101f46004
8036038101906101ef91906117d9565b6107b6565b60405161020191906118b7565b604051809103
90f35b34801561021657600080fd5b50610231600480360381019061022c91906115c2565b61094d
565b60405161023e9190611571565b60405180910390f35b600080600061025885600086610a6956
5b9250509150601660030b82146102a3576040517f08c379a00000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
5181106102b7576102b6611975565b5b60200260200101519250505092915050565b600080600061
5181106102b7576102b6611975565b5b60200260200101519250505092915050565b600080600061
```

016773ff16639b23d3d960e01b8888888860405160
240161030694939291906119b3565b604051602081830303815290604052907bffffffffffffffff
ff19166020820180517bffffffffffffffffffffffff
ff83818316178352505050506040516103709190611a6956
5b600060405180830381855af49150503d80600081146103ab576040519150601f19603f3d011682
016040523d82523d6000602084013e6103b0565b606091505b5091509150816103c15760156103d6
565b808060200190518101906103d59190611ab9565b5b60030b92505050949350505050565b6000
80600061016773ff166315dacbea60e01b88888888
60405160240161042294939291906119b3565b604051602081830303815290604052907bfffff
ff19166020820180517bffffffffffffffff
ff838183161783525050505060405161048c9190
611a69565b6000604051808303816000865af19150503d80600081146104c9576040519150601f19
603f3d011682016040523d82523d6000602084013e6104ce565b606091505b5091509150816104df
5760156104f4565b808060200190518101906104f39190611ab9565b5b60030b9250505094935050
5050565b60008061016773ff1663618dc65e60e01b
858560405160240161053a929190611b30565b604051602081830303815290604052907bfffff
ff19166020820180517bffffffffffffffff
ff83818316178352505050506040516105a49190
611a69565b6000604051808303816000865af19150503d80600081146105e1576040519150601f19
603f3d011682016040523d82523d6000602084013e6105e6565b606091505b50915091507f4af478
0e06fe8cb9df64b0794fa6f01399af979175bb988e35e0e57e594567bc828260405161061c929190
611b7b565b60405180910390a150505050565b60006106368385610be1565b506000610645853086
86610cf9565b9050601660030b811461068d576040517f08c379a0000000000000000000000000
000000000000000000000000000000815260040161068490611c1d565b60405180910390fd5b8091
50509392505050565b600080600061016773ff1663
9b23d3d960e01b888888886040516024016106d594939291906119b3565b60405160208183030381
5290604052907bff1916602082
0180517bff8381831617835250
50505060405161073f9190611a69565b6000604051808303816000865af19150503d806000811461
077c576040519150601f19603f3d011682016040523d82523d6000602084013e610781565b606091
505b5091509150816107925760156107a7565b808060200190518101906107a69190611ab9565b5b
60030b92505050949350505050565b600080600167ffffffffffffffff8111156107d4576107d361
12e3565b5b60405190808252806020026020018201604052801561080d57816020015b6107fa6111
32565b8152602001906001900390816107f25790505b50905061081d6004600130610e17565b8160
008151811061083157610830611975565b5b6020026020010181905250610844611152565b878160
00018190525086816020018190525085816060018190525030816040019073fffffffffffff
ffffffffffff16908173ffffffffffffffffffffffffffffffffffff1681525050
6001816080019015159081151581525050848160a0019060070b908160070b81525050818160e001
8190525060008160c00190151590811515815250506108de3085610e4e565b816101000181905250
6000806108f383610ea6565b91509150601660030b821461093d576040517f08c379a00000000000
00000000000000000000000000000000815260040161093490611caf565b604051
80910390fd5b8094505050505059450505050565b600080600061016773fffffffffffff
ffffffffffff166315dacbea60e01b8888888860405160240161098a94939291906119
b3565b604051602081830303815290604052907bfffffffffffffffffffffffffffffffffffff
ffffffffffff19166020820180517bfffffffffffffffffffffffffffffffffffff
ffffffffffff83818316178352505050506040516109f49190611a69565b600060405180830381855a
f49150503d8060008114610a2f576040519150601f19603f3d011682016040523d82523d60006020
84013e610a34565b606091505b509150915081610a45576015610a5a565b80806020019051810190
610a599190611ab9565b5b60030b92505050949350505050565b600080606060008061016773ffff
ffffffffffffffffffffffffffffffff1663e0f4059a60e01b898989604051602401610aa793
929190611ddb565b604051602081830303815290604052907bfffffffffffffffffffff
ffffffffffff19166020820180517bfffffffffffffffffffff
ffffffffffff8381831617835250505050604051610b119190611a69565b600060405180
8303816000865af19150503d8060008114610b4e576040519150601f19603f3d011682016040523d
82523d6000602084013e610b53565b606091505b509150915081610baf57601560008067fffff
ffffffff811115610b7b57610b7a6112e3565b5b6040519080825280602002602001820160405280
15610ba95781602001602082028036833780820191505090505b50610bc4565b8080602001905181
0190610bc39190611ef1565b5b8260030b925080955081965082975050505050509350935093050
565b600080600061016773fffffffffffff
8686604051602401610c1a929190611f60565b604051602081830303815290604052907bfffff
ffffffffffff19166020820180517bfffffffffffff
ffffffffffff8381831617835250505050604051610c849190
611a69565b6000604051808303816000865af19150503d8060008114610cc1576040519150601f19
603f3d011682016040523d82523d6000602084013e610cc6565b606091505b509150915081610cd7

[illegible]

8868287016112b8565b93505060206116e9868287016112b8565b92505060406116fa86828701611
69c565b9150509250925092565b6000819050919050565b61171781611704565b82525050565b600
0602082019050611732600083018461170e565b92915050565b600067fffffffffffffffff8211156
11753576117526112e3565b5b61175c826112d2565b9050602081019050919050565b600061177c6
1177784611738565b611343565b905082815260208101848484011156117985761179761138f565
b5b6117a38482856113c5565b5093925050565b600082601f8301126117c0576117bf6112cd565
b5b81356117d0848260208601611769565b91505092915050565b600080600080600060a08688031
2156117f5576117f4611265565b5b600086013567fffffffffffffffff8111561181357611812611
26a565b5b61181f888289016117ab565b955050602086013567fffffffffffffffff811156118405
761183f61126a565b5b61184c888289016117ab565b945050604086013567fffffffffffffffff811
11561186d5761186c61126a565b5b611879888289016117ab565b935050606061188a88828901611
69c565b925050608061189b8882890161169c565b9150509295509295909350565b6118b18161128
f565b82525050565b60006020820190506118cc60008301846118a8565b92915050565b600082825
260208201905092915050565b7f4661696c656420746f206d696e74206e6f6e2d66756e6769626c6
520746f6b6560008201527f6e00
00000000602082015250565b600061193f6021836118d2565b915061194a826118e3565b604082019
050919050565b6000602082019050818103600083015261196e81611932565b9050919050565b7f4
e487b7100
0246000fd5b6119ad8161158c565b82525050565b60006080820190506119c860008301876118a85
65b6119d560208301866118a8565b6119e260408301856118a8565b6119ef60608301846119a4565
b959450505050565b600081519050919050565b600081905092915050565b60005b83811015611
a2c578082015181840152602081019050611a11565b600084840152505050565b6000611a43826
119f8565b611a4d8185611a03565b9350611a5d818560208601611a0e565b8084019150509291505
0565b6000611a758284611a38565b915081905092915050565b60008160030b9050919050565b611
a9681611a80565b8114611aa157600080fd5b50565b600081519050611ab381611a8d565b9291505
0565b600060208284031215611acf57611ace611265565b5b6000611add84828501611aa4565b915
05092915050565b600082825260208201905092915050565b6000611b02826119f8565b611b0c818
5611ae6565b9350611b1c818560208601611a0e565b611b25816112d2565b8401915050929150505
65b6000604082019050611b4560008301856118a8565b8181036020830152611b578184611af7565
b90509392505050565b60008115159050919050565b611b7581611b60565b82525050565b6000604
082019050611b906000830185611b6c565b8181036020830152611ba28184611af7565b905093925
05050565b7f4661696c656420746f207472616e73666572206e6f6e2d66756e6769626c652060008
201527f746f6b656e00
250565b6000611c076025836118d2565b9150611c1282611bab565b604082019050919050565b600
06020820190508181036000830152611c3681611bfa565b9050919050565b7f4661696c656420746
f20637265617465206e6f6e2d66756e6769626c6520746f60008201527f6b656e0000000000000000
00
00
2565b9150611ca482611c3d565b604082019050919050565b6000602082019050818103600083015
2611cc881611c8c565b9050919050565b600081519050919050565b6000828252602082019050929
15050565b6000819050602082019050919050565b600082825260208201905092915050565b60006
11d17826119f8565b611d218185611cfb565b9350611d31818560208601611a0e565b611d3a81611
2d2565b840191505092915050565b6000611d518383611d0c565b905092915050565b60006020820
19050919050565b6000611d7182611ccf565b611d7b8185611cda565b935083602082028501611d8
d85611ceb565b8060005b85811015611dc95784840389528151611daa8582611d45565b9450611db
583611d59565b925060208a01995050600181019050611d91565b5082975087955050505050929
15050565b6000606082019050611df060008301866118a8565b611dfd6020830185611562565b818
1036040830152611e0f8184611d66565b9050949350505050565b600081519050611e28816116855
65b92915050565b600067fffffffffffffffff821115611e4957611e486112e3565b5b60208202905
0602081019050919050565b6000611e6d611e6884611e2e565b611343565b9050808382526020820
190506020840283018581115611e9057611e8f61138a565b5b835b81811015611eb95780611ea58
882611e19565b845260208401935050602081019050611e92565b5050509392505050565b6000826
01f830112611ed857611ed76112cd565b5b8151611ee8848260208601611e5a565b9150509291505
0565b600080600060608486031215611f0a57611f09611265565b5b6000611f1886828701611aa45
65b9350506020611f2986828701611e19565b925050604084015167fffffffffffffffff81115611
f4a57611f4961126a565b5b611f5686828701611ec3565b9150509250925092565b6000604082019
050611f7560008301856118a8565b611f8260208301846118a8565b9392505050565b60006080820
19050611f9e60008301876118a8565b611fab60208301866118a8565b611fb860408301856118a85
65b611fc56060830184611562565b959450505050565b600081519050919050565b60008282526
0208201905092915050565b6000611ff582611fce565b611fff8185611fd9565b935061200f81856
0208601611a0e565b612018816112d2565b840191505092915050565b61202c8161128f565b82525
050565b61203b81611b60565b82525050565b61204a8161155565b82525050565b6000815190509
19050565b600082825260208201905092915050565b6000819050602082019050919050565b61208
58161158c565b82525050565b600060a0830160008301516120a36000860182612032565b5060208
301516120b66020860182612023565b50604083015184820360408601526120ce8282611d0c565b9

```
15050606083015184820360608601526120e88282611d0c565b91505060808301516120fd6080860
182612023565b508091505092915050565b6000604083016000830151612120600086018261207c5
65b5060208301518482036020860152612138828261208b565b9150508091505092915050565b600
06121518383612108565b905092915050565b6000602082019050919050565b60006121718261205
0565b61217b818561205b565b93508360208202850161218d8561206c565b8060005b85811015612
1c957848403895281516121aa8582612145565b94506121b583612159565b925060208a019950506
00181019050612191565b508297508795505050505092915050565b6060820160008201516121f
16000850182612041565b5060208201516122046020850182612023565b506040820151612217604
0850182612041565b50505050565b600061016083016000830151848203600086015261223b82826
11fea565b915050602083015184820360208601526122558282611fea565b9150506040830151612
26a6040860182612023565b50606083015184820360608601526122828282611fea565b915050608
08301516122976080860182612032565b5060a08301516122aa60a0860182612041565b5060c0830
1516122bd60c0860182612032565b5060e083015184820360e08601526122d58282612166565b915
0506101008301516122ec6101008601826121db565b508091505092915050565b600060208201905
08181036000830152612311818461221d565b905092915050565b60006123248261126f565b90509
19050565b61233481612319565b811461233f57600080fd5b50565b6000815190506123518161232
b565b92915050565b6000806040838503121561236e5761236d611265565b5b600061237c8582860
1611aa4565b925050602061238d85828601612342565b9150509250929050565b7f4e487b7100000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
264697066735822122063f39734e0d74df5079e24ac5283123560eb80cfa99b53b160e57d06642aa
73664736f6c63430008120033
```

```
{% endtab %}
{% endtabs %}
```

Store your contract on Hedera using `<mark style="color:blue;">ContractCreateFlow()</mark>`. This single call performs `<mark style="color:blue;">FileCreateTransaction()</mark>`, `<mark style="color:blue;">FileAppendTransaction()</mark>`, and `<mark style="color:blue;">ContractCreateTransaction()</mark>` for you. See the difference [here](#).

```
{% tabs %}
{% tab title="Java" %}
```

```
java
// Create contract
ContractCreateFlow createContract = new ContractCreateFlow()
    .setBytecode(bytecode) // Contract bytecode
    .setGas(4000000); // Increase if revert
```

```
TransactionResponse createContractTx = createContract.execute(client);
TransactionReceipt createContractRx = createContractTx.getReceipt(client);
// Get the new contract ID
ContractId newContractId = createContractRx.contractId;
```

```
System.out.println("Contract created with ID: " + newContractId);
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
// Create contract
const createContract = new ContractCreateFlow()
    .setGas(4000000) // Increase if revert
    .setBytecode(bytecode); // Contract bytecode
const createContractTx = await createContract.execute(client);
const createContractRx = await createContractTx.getReceipt(client);
const contractId = createContractRx.contractId;
```

```
console.log("Contract created with ID: ${contractId} \n");
```

```
{% endtab %}
```

```
{% tab title="Go" %}
```

```

go
//Create the transaction
createContract := hedera.NewContractCreateFlow().
    SetGas(4000000).
    SetBytecode([]byte(bytecode))

//Sign the transaction with the client operator key and submit to a Hedera
network
txResponse, err := createContract.Execute(client)
if err != nil {
    panic(err)
}

//Request the receipt of the transaction
receipt, err := txResponse.GetReceipt(client)
if err != nil {
    panic(err)
}

//Get the contract ID
newContractId := receipt.ContractID

fmt.Printf("The new contract ID is %v\n", newContractId)

{% endtab %}
{% endtabs %}

```

3. Execute the Contract to Create an NFT

The parameters you need to specify for this contract call are Name, Symbol, Memo, Maximum Supply, and Expiration. The smart contract "rent" feature is currently NOT enabled. Once enabled in the future, setting an expiration date in seconds is required because entities on Hedera will need to pay "rent" to persist. In this case, the contract entity will pay all NFT auto-renewal fees.

```
{% hint style="warning" %}
```

Note: The expiration must be between 82 and 91 days, specified in seconds. This window will change when HIP-372 replaces HIP-16.

```
{% endhint %}
```

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
// Create NFT using contract
```

```

ContractExecuteTransaction createToken = new ContractExecuteTransaction()
    .setContractId(newContractId) // Contract id
    .setGas(4000000) // Increase if revert
    .setPayableAmount(new Hbar(50)) // Increase if revert
    .setFunction("createNft", new ContractFunctionParameters()
        .addString("Fall Collection") // NFT Name
        .addString("LEAF") // NFT Symbol
        .addString("Just a memo") // NFT Memo
        .addInt64(250) // NFT max supply
        .addInt64(7000000)); // Expiration: Needs to be between 6999999 and
8000000

```

```

TransactionResponse createTokenTx = createToken.execute(client);
TransactionRecord createTokenRx = createTokenTx.getRecord(client);

```

```

String tokenIdSolidityAddr = createTokenRx.contractFunctionResult.getAddress(0);
AccountId tokenId = AccountId.fromSolidityAddress(tokenIdSolidityAddr);

```

```
System.out.println("Token created with ID: " + tokenId);
```

```

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Create NFT from precompile
const createToken = new ContractExecuteTransaction()
    .setContractId(contractId)
    .setGas(4000000) // Increase if revert
    .setPayableAmount(50) // Increase if revert
    .setFunction("createNft",
        new ContractFunctionParameters()
            .addString("Fall Collection") // NFT name
            .addString("LEAF") // NFT symbol
            .addString("Just a memo") // NFT memo
            .addInt64(250) // NFT max supply
            .addInt64(7000000) // Expiration: Needs to be between 6999999 and
8000001
    );
const createTokenTx = await createToken.execute(client);
const createTokenRx = await createTokenTx.getRecord(client);
const tokenIdSolidityAddr = createTokenRx.contractFunctionResult.getAddress(0);
const tokenId = AccountId.fromSolidityAddress(tokenIdSolidityAddr);

console.log(Token created with ID: ${tokenId} \n);

{% endtab %}

{% tab title="Go" %}
go
contractParams := hedera.NewContractFunctionParameters().
    AddString("Fall Collection"). // NFT name
    AddString("LEAF").           // NFT symbol
    AddString("Just a memo").     // NFT memo
    AddInt64(250).                // NFT max supply
    AddInt64(7000000)             // Expiration: Needs to be between 6999999
and 8000001
//Create NFT
createToken, err := hedera.NewContractExecuteTransaction().
    //The contract ID
    SetContractID(newContractId).
    //The max gas
    SetGas(4000000).
    SetPayableAmount(hedera.NewHbar(50)).
    //The contract function to call and parameters
    SetFunction("createNft", contractParams).
    Execute(client)

if err != nil {
    panic(err)
}

//Get the record
txRecord, err := createToken.GetRecord(client)
if err != nil {
    panic(err)
}

//Get transaction status
contractResult, err := txRecord.GetContractExecuteResult()
if err != nil {
    panic(err)
}
tokenIdSolidityAddr := hex.EncodeToString(contractResult.GetAddress(0))

```

```

tokenId, err := hedera.AccountIDFromSolidityAddress(tokenIdSolidityAddr)
if err != nil {
    panic(err)
}

```

```

fmt.Printf("Token created with ID: %v\n", tokenId)

```

```

{% endtab %}
{% endtabs %}

```

4. Execute the Contract to Mint a New NFT

After the token ID is created, you mint each NFT under that ID using the `<mark style="color:blue;">mintNft</mark>` function. For the minting, you must specify the token ID as a Solidity address and the NFT metadata.

Both the NFT image and metadata live in the InterPlanetary File System (IPFS), which provides decentralized storage. The file metadata.json contains the metadata for the NFT. An IPFS URI pointing to the metadata file is used during minting of a new NFT. Notice that the metadata file contains a URI pointing to the NFT image.

```

{% hint style="info" %}
Note: For the latest NFT Token Metadata JSON Schema see HIP-412.
{% endhint %}

```

```

{% tabs %}
{% tab title="Java" %}
java
// Mint NFT
ContractExecuteTransaction mintToken = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(4000000)
    .setMaxTransactionFee(new Hbar(20)) //Use when HBAR is <10 cents
    .setFunction("mintNft", new ContractFunctionParameters()
        .addAddress(tokenIdSolidityAddr) // Token address
        .addByteArray(byteArray)); // Metadata

```

```

TransactionResponse mintTokenTx = mintToken.execute(client);
TransactionRecord mintTokenRx = mintTokenTx.getRecord(client);
// NFT serial number
long serial = mintTokenRx.contractFunctionResult.getInt64(0);

```

```

System.out.println("Minted NFT with serial: " + serial);

```

```

{% endtab %}

```

```

{% tab title="JavaScript" %}
javascript
// IPFS URI
metadata = "ipfs://bafyreie3ichmqul4xa7e6xcy34tylbuq2vf3gnjf7c55trg3b6xyjr4bku/
metadata.json";

```

```

// Mint NFT
const mintToken = new ContractExecuteTransaction()
    .setContractId(contractId)
    .setGas(4000000)
    .setMaxTransactionFee(new hbar(20)) //Use when HBAR is under 10 cents
    .setFunction("mintNft",
        new ContractFunctionParameters()
            .addAddress(tokenIdSolidityAddr) // Token address
            .addByteArray([Buffer.from(metadata)]) // Metadata
    );

```



```

const mintTokenTx = await mintToken.execute(client);
const mintTokenRx = await mintTokenTx.getRecord(client);
const serial = mintTokenRx.contractFunctionResult.getInt64(0);

console.log(Minted NFT with serial: ${serial} \n);

{% endtab %}

{% tab title="Go" %}
go
// ipfs URI
metadata :=
"ipfs://bafyreie3ichmqul4xa7e6xcy34tylbuq2vf3gnjf7c55trg3b6xyjr4bku/
metadata.json"
byteArray := [][]byte{}
byteArray = append(byteArray, []byte(metadata))

// Add token address to params
mintParams, err := hedera.NewContractFunctionParameters().
    AddAddress(tokenIdSolidityAddr)

if err != nil {
    panic(err)
}

// Add metadata to params
mintParams = mintParams.AddByteArray(byteArray)

// Mint NFT
mintToken, err := hedera.NewContractExecuteTransaction().
    //The contract ID
    SetContractID(newContractId).
    //The max gas
    SetGas(4000000).
    //The contract function to call and parameters
    SetFunction("mintNft", mintParams).
    //The max transaction fee. Use when HBAR is under 10 cents
    SetMaxTransactionFee(hedera.HbarFrom(20, hedera.HbarUnits.Hbar)).
    Execute(client)

if err != nil {
    panic(err)
}

//Get the record
mintRecord, err := mintToken.GetRecord(client)
if err != nil {
    panic(err)
}

//Get transaction status
mintResult, err := mintRecord.GetContractExecuteResult()
if err != nil {
    panic(err)
}
serial := mintResult.GetInt64(0)

fmt.Printf("Minted NFT with serial: %v\n", serial)

{% endtab %}
{% endtabs %}

{% tabs %}
{% tab title="metadata.json" %}

```

```

json
{
  "name": "LEAF1.jpg",
  "creator": "Mother Nature",
  "description": "Autumn",
  "type": "image/jpg",
  "format": "none",
  "properties": {
    "city": "Boston",
    "season": "Fall",
    "decade": "20's"
  },
  "image":
  "ipfs://bafybeig35bheyqpi4qlnuljpok54ud753bnp62fe6jit343hv3oxhgmbfm/LEAF1.jpg"
}

{% endtab %}
{% endtabs %}

```

5. Execute the Contract to Transfer the NFT

The NFT is minted to the contract address because the contract is the treasury for the token. Now transfer the NFT to another account or contract address. In this example, you will transfer the NFT to Alice. For the transfer, you must specify the token address and NFT serial number.

The `transferNft` function in the Solidity contract contains a call to an `associateToken` function that will automatically associate Alice to the token ID. This association transaction must be signed using Alice's private key. After signing, Alice will receive the NFT.

```

{% hint style="info" %}
Note: For a more comprehensive explanation of how auto token association works,
check out the Auto Token Associations section here. Reference Hedera Improvement
Proposal: HIP-23
{% endhint %}

```

```

{% tabs %}
{% tab title="Java" %}
java
// Transfer NFT to Alice
ContractExecuteTransaction transferToken = new ContractExecuteTransaction()
    .setContractId(newContractId)
    .setGas(40000000)
    .setFunction("transferNft", new ContractFunctionParameters()
        .addAddress(tokenIdSolidityAddr) // Token id
        .addAddress(aliceId.toSolidityAddress()) // Token receiver (Alice)
        .addInt64(serial)) // Serial number
    .freezeWith(client) // Freeze transaction using client
    .sign(aliceKey); //Sign using Alice Private Key

TransactionResponse transferTokenTx = transferToken.execute(client);
TransactionReceipt transferTokenRx = transferTokenTx.getReceipt(client);

System.out.println("Transfer status: " + transferTokenRx.status);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Transfer NFT to Alice

```

```

const transferToken = await new ContractExecuteTransaction()
    .setContractId(contractId)
    .setGas(40000000)
    .setFunction("transferNft",
        new ContractFunctionParameters()
            .addAddress(tokenIdSolidityAddr) // Token address
            .addAddress(aliceId.toSolidityAddress()) // Token receiver (Alice)
            .addInt64(serial)) // NFT serial number
    .freezeWith(client) // freezing using client
    .sign(aliceKey); // Sign transaction with Alice

const transferTokenTx = await transferToken.execute(client);
const transferTokenRx = await transferTokenTx.getReceipt(client);

console.log(Transfer status: ${transferTokenRx.status} \n);

{% endtab %}

{% tab title="Go" %}
go
// Add token address to params
transferParams, err := hedera.NewContractFunctionParameters().
    AddAddress(tokenIdSolidityAddr)
// Add Alice address to params
transferParams, err =
transferParams.AddAddress(aliceAccountId.ToSolidityAddress())
if err != nil {
    panic(err)
}
transferParams = transferParams.AddInt64(serial)

// Transfer NFT
transferToken, err := hedera.NewContractExecuteTransaction().
    //The contract ID
    SetContractID(newContractId).
    //The max gas
    SetGas(40000000).
    //The contract function to call and parameters
    SetFunction("transferNft", transferParams).
    FreezeWith(client)

if err != nil {
    panic(err)
}

transferSubmit, err := transferToken.Sign(aliceKey).Execute(client)
if err != nil {
    panic(err)
}

//Get the record
transferRecord, err := transferSubmit.GetReceipt(client)
if err != nil {
    panic(err)
}

fmt.Printf("Transfer status: %v\n", transferRecord.Status)

{% endtab %}
{% endtabs %}

```

Code Check 

<details>

<summary>Java</summary>

```
java
package nfthscshts.hedera;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Objects;
import java.util.concurrent.TimeoutException;

import com.hedera.hashgraph.sdk.;
import io.github.cdimascio.dotenv.Dotenv;

public class Deploy {
    private static AccountId accountCreator(PrivateKey pvKey, int iBal, Client
client)
        throws TimeoutException, PrecheckStatusException,
ReceiptStatusException {
        AccountCreateTransaction transaction = new AccountCreateTransaction()
            .setKey(pvKey.getPublicKey())
            .setMaxAutomaticTokenAssociations(10)
            .setInitialBalance(new Hbar(iBal));

        TransactionResponse txResponse = transaction.execute(client);

        TransactionReceipt receipt = txResponse.getReceipt(client);

        return receipt.accountId;
    }

    public static void main(String[] args)
        throws TimeoutException, PrecheckStatusException,
ReceiptStatusException, IOException {
        // ipfs URI
        String metadata =
("ipfs://bafyreie3ichmqul4xa7e6xcy34tylbuq2vf3gnjf7c55trg3b6xyjr4bku/
metadata.json");
        byte[][] byteArray = new byte[1][metadata.length()];
        byteArray[0] = metadata.getBytes();

        AccountId operatorId =
AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("ACCOUNTID")));
        PrivateKey operatorKey =
PrivateKey.fromString(Objects.requireNonNull(Dotenv.load().get("PRIVATEKEY")));

        Client client = Client.forTestnet();
        client.setOperator(operatorId, operatorKey);

        PrivateKey aliceKey = PrivateKey.generateED25519();
        AccountId aliceId = accountCreator(aliceKey, 100, client);
        System.out.print(aliceId);

        String bytecode =
Files.readString(Paths.get("./NFTCreatorsolNFTCreator.bin"));

        // Create contract
        ContractCreateFlow createContract = new ContractCreateFlow()
            .setBytecode(bytecode) // Contract bytecode
            .setGas(40000000); // Increase if revert
```

```

        TransactionResponse createContractTx = createContract.execute(client);
        TransactionReceipt createContractRx =
createContractTx.getReceipt(client);
        // Get the new contract ID
        ContractId newContractId = createContractRx.contractId;

        System.out.println("Contract created with ID: " + newContractId);

        // Create NFT using contract
        ContractExecuteTransaction createToken = new
ContractExecuteTransaction()
            .setContractId(newContractId) // Contract id
            .setGas(40000000) // Increase if revert
            .setPayableAmount(new Hbar(50)) // Increase if revert
            .setFunction("createNft", new ContractFunctionParameters()
                .addString("Fall Collection") // NFT Name
                .addString("LEAF") // NFT Symbol
                .addString("Just a memo") // NFT Memo
                .addInt64(250) // NFT max supply
                .addInt64(70000000)); // Expiration: Needs to be between
69999999 and 80000001

        TransactionResponse createTokenTx = createToken.execute(client);
        TransactionRecord createTokenRx = createTokenTx.getRecord(client);

        String tokenIdSolidityAddr =
createTokenRx.contractFunctionResult.getAddress(0);
        AccountId tokenId = AccountId.fromSolidityAddress(tokenIdSolidityAddr);

        System.out.println("Token created with ID: " + tokenId);

        // Mint NFT
        ContractExecuteTransaction mintToken = new ContractExecuteTransaction()
            .setContractId(newContractId)
            .setGas(40000000)
            .setMaxTransactionFee(new Hbar(20)) // Use when HBAR is <10
cents
            .setFunction("mintNft", new ContractFunctionParameters()
                .addAddress(tokenIdSolidityAddr) // Token address
                .addByteArray(byteArray)); // Metadata

        TransactionResponse mintTokenTx = mintToken.execute(client);
        TransactionRecord mintTokenRx = mintTokenTx.getRecord(client);
        // NFT serial number
        long serial = mintTokenRx.contractFunctionResult.getInt64(0);

        System.out.println("Minted NFT with serial: " + serial);

        // Transfer NFT to Alice
        ContractExecuteTransaction transferToken = new
ContractExecuteTransaction()
            .setContractId(newContractId)
            .setGas(40000000)
            .setFunction("transferNft", new ContractFunctionParameters()
                .addAddress(tokenIdSolidityAddr) // Token id
                .addAddress(aliceId.toSolidityAddress()) // Token
receiver (Alice)
                .addInt64(serial)) // Serial number
            .freezeWith(client) // Freeze transaction using client
            .sign(aliceKey); // Sign using Alice Private Key

        TransactionResponse transferTokenTx = transferToken.execute(client);
        TransactionReceipt transferTokenRx = transferTokenTx.getReceipt(client);

```

```

        System.out.println("Transfer status: " + transferTokenRx.status);
    }
}

</details>

<details>

<summary>JavaScript</summary>

javascript
console.clear();
require("dotenv").config();
const fs = require("fs");
const {
    AccountId,
    PrivateKey,
    Client,
    ContractCreateFlow,
    ContractExecuteTransaction,
    ContractFunctionParameters,
    AccountCreateTransaction,
    Hbar,
} = require("@hashgraph/sdk");

// ipfs URI
metadata =
    "ipfs://bafyreie3ichmqul4xa7e6xcy34tylbuq2vf3gnjf7c55trg3b6xyjr4bku/
metadata.json";

const operatorKey = PrivateKey.fromString(process.env.MYPRIVATEKEY);
const operatorId = AccountId.fromString(process.env.MYACCOUNTID);

const client = Client.forTestnet().setOperator(operatorId, operatorKey);

// Account creation function
async function accountCreator(pvKey, iBal) {
    const response = await new AccountCreateTransaction()
        .setInitialBalance(new Hbar(iBal))
        .setKey(pvKey.publicKey)
        .setMaxAutomaticTokenAssociations(10)
        .execute(client);
    const receipt = await response.getReceipt(client);
    return receipt.accountId;
}

const main = async () => {
    // Init Alice account
    const aliceKey = PrivateKey.generateED25519();
    const aliceId = await accountCreator(aliceKey, 100);

    const bytecode = fs.readFileSync("./binaries/NFTCreatorsolNFTCreator.bin");

    // Create contract
    const createContract = new ContractCreateFlow()
        .setGas(4000000) // Increase if revert
        .setBytecode(bytecode); // Contract bytecode
    const createContractTx = await createContract.execute(client);
    const createContractRx = await createContractTx.getReceipt(client);
    const contractId = createContractRx.contractId;

```

```

console.log(Contract created with ID: ${contractId} \n);

// Create NFT from precompile
const createToken = new ContractExecuteTransaction()
  .setContractId(contractId)
  .setGas(40000000) // Increase if revert
  .setPayableAmount(50) // Increase if revert
  .setFunction(
    "createNft",
    new ContractFunctionParameters()
      .addString("Fall Collection") // NFT name
      .addString("LEAF") // NFT symbol
      .addString("Just a memo") // NFT memo
      .addInt64(250) // NFT max supply
      .addInt64(70000000) // Expiration: Needs to be between 69999999 and
80000001
  );
const createTokenTx = await createToken.execute(client);
const createTokenRx = await createTokenTx.getRecord(client);
const tokenIdSolidityAddr =
  createTokenRx.contractFunctionResult.getAddress(0);
const tokenId = AccountId.fromSolidityAddress(tokenIdSolidityAddr);

console.log(Token created with ID: ${tokenId} \n);

// Mint NFT
const mintToken = new ContractExecuteTransaction()
  .setContractId(contractId)
  .setGas(40000000)
  .setMaxTransactionFee(new Hbar(20)) //Use when HBAR is under 10 cents
  .setFunction(
    "mintNft",
    new ContractFunctionParameters()
      .addAddress(tokenIdSolidityAddr) // Token address
      .addByteArray([Buffer.from(metadata)]) // Metadata
  );
const mintTokenTx = await mintToken.execute(client);
const mintTokenRx = await mintTokenTx.getRecord(client);
const serial = mintTokenRx.contractFunctionResult.getInt64(0);

console.log(Minted NFT with serial: ${serial} \n);

// Transfer NFT to Alice
const transferToken = await new ContractExecuteTransaction()
  .setContractId(contractId)
  .setGas(40000000)
  .setFunction(
    "transferNft",
    new ContractFunctionParameters()
      .addAddress(tokenIdSolidityAddr) // Token address
      .addAddress(aliceId.toSolidityAddress()) // Token receiver (Alice)
      .addInt64(serial)
  ) // NFT serial number
  .freezeWith(client) // freezing using client
  .sign(aliceKey); // Sign transaction with Alice
const transferTokenTx = await transferToken.execute(client);
const transferTokenRx = await transferTokenTx.getReceipt(client);

console.log(Transfer status: ${transferTokenRx.status} \n);
};

main();

```

</details>

<details>

<summary>Go</summary>

```
go
package main

import (
    "encoding/hex"
    "fmt"
    "io/ioutil"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {
    godotenv.Load("../.env")

    metadata :=
"ipfs://bafyreie3ichmqul4xa7e6xcy34tylbuq2vf3gnjf7c55trg3b6xyjr4bku/
metadata.json"
    byteArray := [][]byte{}
    byteArray = append(byteArray, []byte(metadata))

    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%v\n", err))
    }

    //Grab your testnet account ID and private key from the .env file
    operatorId, err := hedera.AccountIDFromString(os.Getenv("ACCOUNTID"))
    if err != nil {
        panic(err)
    }

    operatorKey, err := hedera.PrivateKeyFromString(os.Getenv("PRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    //Create your testnet client
    client := hedera.ClientForTestnet()
    client.SetOperator(operatorId, operatorKey)

    aliceKey, err := hedera.PrivateKeyGenerateEd25519()
    if err != nil {
        panic(err)
    }

    //Create the transaction
    accountCreate := hedera.NewAccountCreateTransaction().
        SetKey(aliceKey.PublicKey()).
        SetMaxAutomaticTokenAssociations(10).
        SetInitialBalance(hedera.NewHbar(100))

    //Sign the transaction with the client operator private key and submit to
a Hedera network
    accountCreateSubmit, err := accountCreate.Execute(client)
    if err != nil {
```



```

        panic(err)
    }

    //Request the receipt of the transaction
    accountCreateReceipt, err := accountCreateSubmit.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    //Get the account ID
    aliceAccountId := accountCreateReceipt.AccountID

    // Make sure to close client after running
    defer func() {
        err = client.Close()
        if err != nil {
            println(err.Error(), ": error closing client")
            return
        }
    }()

    // Read bytecode
    bytecode, err := ioutil.ReadFile("./NFTCreatorsolNFTCreator.bin")
    if err != nil {
        println(err.Error(), ": error reading bytecode")
        return
    }

    //Create the transaction
    createContract := hedera.NewContractCreateFlow().
        SetGas(4000000).
        SetBytecode([]byte(bytecode))

    //Sign the transaction with the client operator key and submit to a Hedera
network    txResponse, err := createContract.Execute(client)
    if err != nil {
        panic(err)
    }

    //Request the receipt of the transaction
    receipt, err := txResponse.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    //Get the contract ID
    newContractId := receipt.ContractID

    fmt.Printf("The new contract ID is %v\n", newContractId)

    contractParams := hedera.NewContractFunctionParameters().
        AddString("Fall Collection"). // NFT name
        AddString("LEAF").           // NFT symbol
        AddString("Just a memo").     // NFT memo
        AddInt64(250).                // NFT max supply
        AddInt64(7000000)             // Expiration: Needs to be between
6999999 and 8000001
    //Create NFT
    createToken, err := hedera.NewContractExecuteTransaction().
        //The contract ID
        SetContractID(newContractId).
        //The max gas
        SetGas(4000000).

```

```

        SetPayableAmount(hedera.NewHbar(50)).
        //The contract function to call and parameters
        SetFunction("createNft", contractParams).
        Execute(client)

    if err != nil {
        panic(err)
    }

    //Get the record
    txRecord, err := createToken.GetRecord(client)
    if err != nil {
        panic(err)
    }

    //Get transaction status
    contractResult, err := txRecord.GetContractExecuteResult()
    if err != nil {
        panic(err)
    }
    tokenIdSolidityAddr := hex.EncodeToString(contractResult.GetAddress(0))

    tokenId, err := hedera.AccountIDFromSolidityAddress(tokenIdSolidityAddr)
    if err != nil {
        panic(err)
    }

    fmt.Printf("Token created with ID: %v\n", tokenId)

    // Add token address to params
    mintParams, err := hedera.NewContractFunctionParameters().
        AddAddress(tokenIdSolidityAddr)

    if err != nil {
        panic(err)
    }

    // Add metadata to params
    mintParams = mintParams.AddByteArray(byteArray)

    // Mint NFT
    mintToken, err := hedera.NewContractExecuteTransaction().
        //The contract ID
        SetContractID(newContractId).
        //The max gas
        SetGas(10000000).
        //The contract function to call and parameters
        SetFunction("mintNft", mintParams).
        //The max transaction fee. Use when HBAR is under 10 cents
        SetMaxTransactionFee(hedera.HbarFrom(20, hedera.HbarUnits.Hbar)).
        Execute(client)

    if err != nil {
        panic(err)
    }

    //Get the record
    mintRecord, err := mintToken.GetRecord(client)
    if err != nil {
        panic(err)
    }

    //Get transaction status
    mintResult, err := mintRecord.GetContractExecuteResult()

```

```

    if err != nil {
        panic(err)
    }
    serial := mintResult.GetInt64(0)

    fmt.Printf("Minted NFT with serial: %v\n", serial)

    // Add token address to params
    transferParams, err := hedera.NewContractFunctionParameters().
        AddAddress(tokenIdSolidityAddr)
    // Add Alice address to params
    transferParams, err =
transferParams.AddAddress(aliceAccountId.ToSolidityAddress())
    if err != nil {
        panic(err)
    }
    transferParams = transferParams.AddInt64(serial)

    // Transfer NFT
    transferToken, err := hedera.NewContractExecuteTransaction().
        //The contract ID
        SetContractID(newContractId).
        //The max gas
        SetGas(4000000).
        //The contract function to call and parameters
        SetFunction("transferNft", transferParams).
        FreezeWith(client)

    if err != nil {
        panic(err)
    }

    transferSubmit, err := transferToken.Sign(aliceKey).Execute(client)
    if err != nil {
        panic(err)
    }

    //Get the record
    transferRecord, err := transferSubmit.GetReceipt(client)
    if err != nil {
        panic(err)
    }

    fmt.Printf("Transfer status: %v\n", transferRecord.Status)
}

```

</details>

```

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

```

create-and-transfer-your-first-fungible-token.md:

Create and Transfer Your First Fungible Token

Summary

Fungible tokens share a single set of properties and have interchangeable value with one another. Use cases for fungible tokens include applications like stablecoins, in-game rewards systems, crypto tokens, loyalty program points, and much more.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

Get a Hedera testnet account.
Set up your environment here.

1. Create a Fungible Token

Use `TokenCreateTransaction()` to create a fungible token and configure its properties. At a minimum, this constructor requires setting a name, symbol, and treasury account ID. All other fields are optional, so if they're not specified then default values are used. For instance, not specifying an admin key, makes a token immutable (can't change or add properties); not specifying a supply key, makes a token supply fixed (can't mint new or burn existing tokens); not specifying a token type, makes a token fungible.

After submitting the transaction to the Hedera network, you can obtain the new token ID by requesting the receipt.

Unlike NFTs, fungible tokens do not require the decimals and initial supply to be set to zero during creation. In this case, we set the initial supply to 100 units, which is shown in the code as 10000 to account to 2 decimals.

```
{% tabs %}
{% tab title="Java" %}
java
// CREATE FUNGIBLE TOKEN (STABLECOIN)
TokenCreateTransaction tokenCreateTx = new TokenCreateTransaction()
    .setTokenName("USD Bar")
    .setTokenSymbol("USDB")
    .setTokenType(TokenType.FUNGIBLECOMMON)
    .setDecimals(2)
    .setInitialSupply(10000)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.INFINITE)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

//SIGN WITH TREASURY KEY
TokenCreateTransaction tokenCreateSign = tokenCreateTx.sign(treasuryKey);

//SUBMIT THE TRANSACTION
TransactionResponse tokenCreateSubmit = tokenCreateSign.execute(client);

//GET THE TRANSACTION RECEIPT
TransactionReceipt tokenCreateRx = tokenCreateSubmit.getReceipt(client);

//GET THE TOKEN ID
TokenId tokenId = tokenCreateRx.tokenId;

//LOG THE TOKEN ID TO THE CONSOLE
System.out.println("Created token with ID: " +tokenId);

{% endtab %}
```

```

{% tab title="JavaScript" %}
javascript
// CREATE FUNGIBLE TOKEN (STABLECOIN)
let tokenCreateTx = await new TokenCreateTransaction()
    .setTokenName("USD Bar")
    .setTokenSymbol("USDB")
    .setTokenType(TokenType.FungibleCommon)
    .setDecimals(2)
    .setInitialSupply(10000)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.Infinite)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

//SIGN WITH TREASURY KEY
let tokenCreateSign = await tokenCreateTx.sign(treasuryKey);

//SUBMIT THE TRANSACTION
let tokenCreateSubmit = await tokenCreateSign.execute(client);

//GET THE TRANSACTION RECEIPT
let tokenCreateRx = await tokenCreateSubmit.getReceipt(client);

//GET THE TOKEN ID
let tokenId = tokenCreateRx.tokenId;

//LOG THE TOKEN ID TO THE CONSOLE
console.log(- Created token with ID: ${tokenId} \n);

{% endtab %}

{% tab title="Go" %}
go
//CREATE FUNGIBLE TOKEN (STABLECOIN)
tokenCreateTx, err := hedera.NewTokenCreateTransaction().
    SetTokenName("USD Bar").
    SetTokenSymbol("USDB").
    SetTokenType(hedera.TokenTypeFungibleCommon).
    SetDecimals(2).
    SetInitialSupply(10000).
    SetTreasuryAccountID(treasuryAccountId).
    SetSupplyType(hedera.TokenSupplyTypeInfinite).
    SetSupplyKey(supplyKey).
    FreezeWith(client)

//SIGN WITH TREASURY KEY
tokenCreateSign := tokenCreateTx.Sign(treasuryKey)

//SUBMIT THE TRANSACTION
tokenCreateSubmit, err := tokenCreateSign.Execute(client)

//GET THE TRANSACTION RECEIPT
tokenCreateRx, err := tokenCreateSubmit.GetReceipt(client)

//GET THE TOKEN ID
tokenId := tokenCreateRx.TokenID

//LOG THE TOKEN ID TO THE CONSOLE
fmt.Println("Created fungible token with token ID", tokenId)

{% endtab %}
{% endtabs %}

```

2. Associate User Accounts with Token

Before an account that is not the treasury for a token can receive or send this specific token ID, the account must become "associated" with the token.

```
{% tabs %}
{% tab title="Java" %}
java
// TOKEN ASSOCIATION WITH ALICE'S ACCOUNT
TokenAssociateTransaction associateAliceTx = new TokenAssociateTransaction()
    .setAccountId(aliceAccountId)
    .setTokenIds(Collections.singletonList(tokenId))
    .freezeWith(client)
    .sign(aliceKey);

//SUBMIT THE TRANSACTION
TransactionResponse associateAliceTxSubmit = associateAliceTx.execute(client);

//GET THE RECEIPT OF THE TRANSACTION
TransactionReceipt associateAliceRx = associateAliceTxSubmit.getReceipt(client);

//LOG THE TRANSACTION STATUS
System.out.println("Token association with Alice's account: "
+associateAliceRx.status);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// TOKEN ASSOCIATION WITH ALICE'S ACCOUNT
let associateAliceTx = await new TokenAssociateTransaction()
    .setAccountId(aliceId)
    .setTokenIds([tokenId])
    .freezeWith(client)
    .sign(aliceKey);

//SUBMIT THE TRANSACTION
let associateAliceTxSubmit = await associateAliceTx.execute(client);

//GET THE RECEIPT OF THE TRANSACTION
let associateAliceRx = await associateAliceTxSubmit.getReceipt(client);

//LOG THE TRANSACTION STATUS
console.log(- Token association with Alice's account: ${associateAliceRx.status}
\n);

{% endtab %}

{% tab title="Go" %}
go
//TOKEN ASSOCIATION WITH ALICE'S ACCOUNT
associateAliceTx, err := hedera.NewTokenAssociateTransaction().
    SetAccountId(aliceAccountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

//SIGN WITH ALICE'S KEY TO AUTHORIZE THE ASSOCIATION
signTx := associateAliceTx.Sign(aliceKey)

//SUBMIT THE TRANSACTION
associateAliceTxSubmit, err := signTx.Execute(client)

//GET THE RECEIPT OF THE TRANSACTION
```

```
associateAliceRx, err := associateAliceTxSubmit.GetReceipt(client)
```

```
//LOG THE TRANSACTION STATUS
```

```
fmt.Println("Non-fungible token association with Alice's account:",  
associateAliceRx.Status)
```

```
{% endtab %}
```

```
{% endtabs %}
```

3. Transfer the Token

Now, transfer 25 units of the token from the Treasury to Alice and check the account balances before and after the transfer.

```
{% tabs %}
```

```
{% tab title="Java" %}
```

```
java
```

```
//BALANCE CHECK
```

```
AccountBalance balanceCheckTreasury = new
```

```
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
```

```
System.out.println(" Treasury balance: " +balanceCheckTreasury.tokens + " units  
of token ID" +tokenId);
```

```
AccountBalance balanceCheckAlice = new
```

```
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
```

```
System.out.println("Alice's balance: " + balanceCheckAlice.tokens + " units of  
token ID " + tokenId);
```

```
//TRANSFER STABLECOIN FROM TREASURY TO ALICE
```

```
TransferTransaction tokenTransferTx = new TransferTransaction()
```

```
    .addTokenTransfer(tokenId, treasuryId, -5)
```

```
    .addTokenTransfer(tokenId, aliceAccountId, 5)
```

```
    .freezeWith(client)
```

```
    .sign(treasuryKey);
```

```
//SUBMIT THE TRANSACTION
```

```
TransactionResponse tokenTransferSubmit = tokenTransferTx.execute(client);
```

```
//GET THE RECEIPT OF THE TRANSACTION
```

```
TransactionReceipt tokenTransferRx = tokenTransferSubmit.getReceipt(client);
```

```
//LOG THE TRANSACTION STATUS
```

```
System.out.println("Stablecoin transfer from Treasury to Alice: " +  
tokenTransferRx.status );
```

```
//BALANCE CHECK
```

```
AccountBalance balanceCheckTreasury2 = new
```

```
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
```

```
System.out.println("Treasury balance " + balanceCheckTreasury2.tokens + " units  
of token ID " + tokenId);
```

```
AccountBalance balanceCheckAlice2 = new
```

```
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
```

```
System.out.println("Alice's balance: " +balanceCheckAlice2.tokens + " units of  
token ID " + tokenId);
```

```
{% endtab %}
```

```
{% tab title="JavaScript" %}
```

```
javascript
```

```
//BALANCE CHECK
```

```
var balanceCheckTx = await new
```

```
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
```

```

console.log(- Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
console.log(- Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});

// TRANSFER STABLECOIN FROM TREASURY TO ALICE
let tokenTransferTx = await new TransferTransaction()
    .addTokenTransfer(tokenId, treasuryId, -5)
    .addTokenTransfer(tokenId, aliceId, 5)
    .freezeWith(client)
    .sign(treasuryKey);

//SUBMIT THE TRANSACTION
let tokenTransferSubmit = await tokenTransferTx.execute(client);

//GET THE RECEIPT OF THE TRANSACTION
let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);

//LOG THE TRANSACTION STATUS
console.log(\n- Stablecoin transfer from Treasury to Alice: $
{tokenTransferRx.status} \n);

// BALANCE CHECK
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
console.log(- Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
console.log(- Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});

{% endtab %}

{% tab title="Go" %}
go
//CHECK THE BALANCE BEFORE THE TRANSFER FROM THE TREASURY ACCOUNT
balanceCheckTreasury, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
fmt.Println("Treasury balance:", balanceCheckTreasury.Tokens, "units of token
ID", tokenId)

//CHECK THE BALANCE BEFORE THE TRANSFER FOR ALICE'S ACCOUNT
balanceCheckAlice, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
fmt.Println("Alice's balance:", balanceCheckAlice.Tokens, "units of token ID",
tokenId)

//TRANSFER THE STABLECOIN TO ALICE
tokenTransferTx, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId, treasuryAccountId, -5).
    AddTokenTransfer(tokenId, aliceAccountId, 5).
    FreezeWith(client)

//SIGN WITH THE TREASURY KEY TO AUTHORIZE THE TRANSFER
signTransferTx := tokenTransferTx.Sign(treasuryKey)

//SUBMIT THE TRANSACTION

```



```

tokenTransferSubmit, err := signTransferTx.Execute(client)

//GET THE TRANSACTION RECEIPT
tokenTransferRx, err := tokenTransferSubmit.GetReceipt(client)


fmt.Println("Token transfer from Treasury to Alice:", tokenTransferRx.Status)

//CHECK THE BALANCE AFTER THE TRANSFER FOR THE TREASURY ACCOUNT
balanceCheckTreasury2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
fmt.Println("Treasury balance:", balanceCheckTreasury2.Tokens, "units of token",
tokenId)

//CHECK THE BALANCE AFTER THE TRANSFER FOR ALICE'S ACCOUNT
balanceCheckAlice2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
gofmt.Println("Alice's balance:", balanceCheckAlice2.Tokens, "units of token",
tokenId)

{% endtab %}
{% endtabs %}

```

Code Check 

<details>

<summary>Java</summary>

```

java
import com.hedera.hashgraph.sdk.;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.Collections;
import java.util.Objects;
import java.util.concurrent.TimeoutException;

public class CreateFungibleTutorial {
    public static void main(String[] args) throws TimeoutException,
PrecheckStatusException, ReceiptStatusException {

        //Grab your Hedera testnet account ID and private key
        AccountId myAccountId =
AccountId.fromString(Objects.requireNonNull(Dotenv.load().get("MYACCOUNTID")));
        PrivateKey myPrivateKey =
PrivateKey.fromString(Objects.requireNonNull(Dotenv.load().get("MYPRIVATEKEY")))
;

        //Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        //Treasury Key
        PrivateKey treasuryKey = PrivateKey.generate();
        PublicKey treasuryPublicKey = treasuryKey.getPublicKey();

        //Create treasury account
        TransactionResponse treasuryAccount = new AccountCreateTransaction()
            .setKey(treasuryPublicKey)
            .setInitialBalance(new Hbar(5))
            .execute(client);

        AccountId treasuryId = treasuryAccount.getReceipt(client).accountId;

```

```

//Alice Key
PrivateKey aliceKey = PrivateKey.generate();
PublicKey alicePublicKey = aliceKey.getPublicKey();

//Create Alice's account
TransactionResponse aliceAccount = new AccountCreateTransaction()
    .setKey(alicePublicKey)
    .setInitialBalance(new Hbar(5))
    .execute(client);

AccountId aliceAccountId = aliceAccount.getReceipt(client).accountId;

//Alice Key
PrivateKey supplyKey = PrivateKey.generate();
PublicKey supplyPublicKey = supplyKey.getPublicKey();

// CREATE FUNGIBLE TOKEN (STABLECOIN)
TokenCreateTransaction tokenCreateTx = new TokenCreateTransaction()
    .setTokenName("USD Bar")
    .setTokenSymbol("USDB")
    .setTokenType(TokenType.FUNGIBLECOMMON)
    .setDecimals(2)
    .setInitialSupply(10000)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.INFINITE)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

//Sign with the treasury key
TokenCreateTransaction tokenCreateSign =
tokenCreateTx.sign(treasuryKey);

//Submit the transaction
TransactionResponse tokenCreateSubmit = tokenCreateSign.execute(client);

//Get the transaction receipt
TransactionReceipt tokenCreateRx = tokenCreateSubmit.getReceipt(client);

//Get the token ID
TokenId tokenId = tokenCreateRx.tokenId;

//Log the token ID to the console
System.out.println("Created token with ID: " +tokenId);

// TOKEN ASSOCIATION WITH ALICE'S ACCOUNT
TokenAssociateTransaction associateAliceTx = new
TokenAssociateTransaction()
    .setAccountId(aliceAccountId)
    .setTokenIds(Collections.singletonList(tokenId))
    .freezeWith(client)
    .sign(aliceKey);

//Submit the transaction
TransactionResponse associateAliceTxSubmit =
associateAliceTx.execute(client);

//Get the receipt of the transaction
TransactionReceipt associateAliceRx =
associateAliceTxSubmit.getReceipt(client);

//Get the transaction status
System.out.println("Token association with Alice's account: "
+associateAliceRx.status);

```

```

        // BALANCE CHECK
        AccountBalance balanceCheckTreasury = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
        System.out.println(" Treasury balance: " +balanceCheckTreasury.tokens +
" units of token ID" +tokenId);
        AccountBalance balanceCheckAlice = new
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
        System.out.println("Alice's balance: " + balanceCheckAlice.tokens + "
units of token ID " + tokenId);

        // TRANSFER STABLECOIN FROM TREASURY TO ALICE
        TransferTransaction tokenTransferTx = new TransferTransaction()
            .addTokenTransfer(tokenId, treasuryId, -5)
            .addTokenTransfer(tokenId, aliceAccountId, 5)
            .freezeWith(client)
            .sign(treasuryKey);

        //SUBMIT THE TRANSACTION
        TransactionResponse tokenTransferSubmit =
tokenTransferTx.execute(client);

        //GET THE RECEIPT OF THE TRANSACTION
        TransactionReceipt tokenTransferRx =
tokenTransferSubmit.getReceipt(client);

        //LOG THE TRANSACTION STATUS
        System.out.println("Stablecoin transfer from Treasury to Alice: " +
tokenTransferRx.status );

        // BALANCE CHECK
        AccountBalance balanceCheckTreasury2 = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
        System.out.println("Treasury balance " + balanceCheckTreasury2.tokens +
" units of token ID " + tokenId);
        AccountBalance balanceCheckAlice2 = new
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
        System.out.println("Alice's balance: " +balanceCheckAlice2.tokens + "
units of token ID " + tokenId);
    }
}

```

</details>

<details>

<summary>JavaScript</summary>

```

javascript
console.clear();
require("dotenv").config();
const {
    AccountId,
    PrivateKey,
    Client,
    TokenCreateTransaction,
    TokenType,
    TokenSupplyType,
    TransferTransaction,
    AccountBalanceQuery,
    TokenAssociateTransaction,
} = require("@hashgraph/sdk");

```

```

// Configure accounts and client, and generate needed keys
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromString(process.env.OPERATORPVKEY);
const treasuryId = AccountId.fromString(process.env.TREASURYID);
const treasuryKey = PrivateKey.fromString(process.env.TREASURYPVKEY);
const aliceId = AccountId.fromString(process.env.ALICEID);
const aliceKey = PrivateKey.fromString(process.env.ALICEPVKEY);

const client = Client.forTestnet().setOperator(operatorId, operatorKey);

const supplyKey = PrivateKey.generate();

async function createFungibleToken() {
  //CREATE FUNGIBLE TOKEN (STABLECOIN)
  let tokenCreateTx = await new TokenCreateTransaction()
    .setTokenName("USD Bar")
    .setTokenSymbol("USDB")
    .setTokenType(TokenType.FungibleCommon)
    .setDecimals(2)
    .setInitialSupply(10000)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.Infinite)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

  let tokenCreateSign = await tokenCreateTx.sign(treasuryKey);
  let tokenCreateSubmit = await tokenCreateSign.execute(client);
  let tokenCreateRx = await tokenCreateSubmit.getReceipt(client);
  let tokenId = tokenCreateRx.tokenId;
  console.log(- Created token with ID: ${tokenId} \n);

  //TOKEN ASSOCIATION WITH ALICE's ACCOUNT
  let associateAliceTx = await new TokenAssociateTransaction()
    .setAccountId(aliceId)
    .setTokenIds([tokenId])
    .freezeWith(client)
    .sign(aliceKey);
  let associateAliceTxSubmit = await associateAliceTx.execute(client);
  let associateAliceRx = await associateAliceTxSubmit.getReceipt(client);
  console.log(- Token association with Alice's account: $
{associateAliceRx.status} \n);

  //BALANCE CHECK
  var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
  console.log(- Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});
  var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
  console.log(- Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});

  //TRANSFER STABLECOIN FROM TREASURY TO ALICE
  let tokenTransferTx = await new TransferTransaction()
    .addTokenTransfer(tokenId, treasuryId, -5)
    .addTokenTransfer(tokenId, aliceId, 5)
    .freezeWith(client)
    .sign(treasuryKey);
  let tokenTransferSubmit = await tokenTransferTx.execute(client);
  let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);
  console.log(\n- Stablecoin transfer from Treasury to Alice: $

```

```

{tokenTransferRx.status} \n);

    //BALANCE CHECK
    var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
    console.log(- Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});
    var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
    console.log(- Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} units of token ID $
{tokenId});
}
createFungibleToken();

```

</details>

<details>

<summary>Go</summary>

```

go
package main

import (
    "fmt"
    "os"

    "github.com/hashgraph/hedera-sdk-go/v2"
    "github.com/joho/godotenv"
)

func main() {

    //LOADS THE .ENV FILE AND THROWS AN EROOR IF IT CANNOT LOAD THE VARIABLES
    err := godotenv.Load(".env")
    if err != nil {
        panic(fmt.Errorf("Unable to load environment variables from .env
file. Error:\n%v\n", err))
    }

    //GRAB YOUR TESTNET ACCOUNT ID AND KEY FROMZ THE .ENV FILE
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))
    if err != nil {
        panic(err)
    }

    myPrivateKey, err :=
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))
    if err != nil {
        panic(err)
    }

    //PRINT ACCOUNT ID AND KEY TO MAKE SURE THERE WASN'T AN ERROR READING FROM
THE .ENV FILE
    fmt.Printf("The account ID is = %v\n", myAccountId)
    fmt.Printf("The private key is = %v\n", myPrivateKey)

    //CREATE TESTNET CLIENT
    client := hedera.ClientForTestnet()
    client.SetOperator(myAccountId, myPrivateKey)

```

```

//CREATE TREASURY KEY
treasuryKey, err := hedera.GeneratePrivateKey()
treasuryPublicKey := treasuryKey.PublicKey()

//CREATE TREASURY ACCOUNT
treasuryAccount, err := hedera.NewAccountCreateTransaction().
    SetKey(treasuryPublicKey).
    SetInitialBalance(hedera.NewHbar(5)).
    Execute(client)

//GET THE RECEIPT OF THE TRANSACTION
receipt, err := treasuryAccount.GetReceipt(client)

//GET THE ACCOUNT ID
treasuryAccountId := receipt.AccountID

//ALICE'S KEY
aliceKey, err := hedera.GeneratePrivateKey()
alicePublicKey := aliceKey.PublicKey()

//CREATE ALICE'S ACCOUNT
aliceAccount, err := hedera.NewAccountCreateTransaction().
    SetKey(alicePublicKey).
    SetInitialBalance(hedera.NewHbar(5)).
    Execute(client)

//GET THE RECEIPT OF THE TRANSACTION
receipt2, err := aliceAccount.GetReceipt(client)

//GET ALICE'S ACCOUNT ID
aliceAccountId := receipt2.AccountID

//CREATE SUPPLY KEY
supplyKey, err := hedera.GeneratePrivateKey()

//CREATE FUNGIBLE TOKEN (STABLECOIN)
tokenCreateTx, err := hedera.NewTokenCreateTransaction().
    SetTokenName("USD Bar").
    SetTokenSymbol("USDB").
    SetTokenType(hedera.TokenTypeFungibleCommon).
    SetDecimals(2).
    SetInitialSupply(10000).
    SetTreasuryAccountId(treasuryAccountId).
    SetSupplyType(hedera.TokenSupplyTypeInfinite).
    SetSupplyKey(supplyKey).
    FreezeWith(client)

//SIGN WITH TREASURY KEY
tokenCreateSign := tokenCreateTx.Sign(treasuryKey)

//SUBMIT THE TRANSACTION
tokenCreateSubmit, err := tokenCreateSign.Execute(client)

//GET THE TRANSACTION RECEIPT
tokenCreateRx, err := tokenCreateSubmit.GetReceipt(client)

//GET THE TOKEN ID
tokenId := tokenCreateRx.TokenID

//LOG THE TOKEN ID TO THE CONSOLE
fmt.Println("Created fungible token with token ID", tokenId)

//TOKEN ASSOCIATION WITH ALICE'S ACCOUNT
associateAliceTx, err := hedera.NewTokenAssociateTransaction().

```

```

        SetAccountID(aliceAccountId).
        SetTokenIDs(tokenId).
        FreezeWith(client)

//SIGN WITH ALICE'S KEY TO AUTHORIZE THE ASSOCIATION
signTx := associateAliceTx.Sign(aliceKey)

//SUBMIT THE TRANSACTION
associateAliceTxSubmit, err := signTx.Execute(client)

//GET THE RECEIPT OF THE TRANSACTION
associateAliceRx, err := associateAliceTxSubmit.GetReceipt(client)

//LOG THE TRANSACTION STATUS
fmt.Println("STABLECOIN token association with Alice's account:",
associateAliceRx.Status)

//Check the balance before the transfer for the treasury account
balanceCheckTreasury, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
fmt.Println("Treasury balance:", balanceCheckTreasury.Tokens, "units of
token ID", tokenId)

//Check the balance before the transfer for Alice's account
balanceCheckAlice, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
fmt.Println("Alice's balance:", balanceCheckAlice.Tokens, "units of token
ID", tokenId)

//Transfer the STABLECOIN from treasury to Alice
tokenTransferTx, err := hedera.NewTransferTransaction().
    AddTokenTransfer(tokenId, treasuryAccountId, -5).
    AddTokenTransfer(tokenId, aliceAccountId, 5).
    FreezeWith(client)

//SIGN WITH THE TREASURY KEY TO AUTHORIZE THE TRANSFER
signTransferTx := tokenTransferTx.Sign(treasuryKey)

//SUBMIT THE TRANSACTION
tokenTransferSubmit, err := signTransferTx.Execute(client)

//GET THE TRANSACTION RECEIPT
tokenTransferRx, err := tokenTransferSubmit.GetReceipt(client)

fmt.Println("Token transfer from Treasury to Alice:",
tokenTransferRx.Status)

//CHECK THE BALANCE AFTER THE TRANSFER FOR THE TREASURY ACCOUNT
balanceCheckTreasury2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
fmt.Println("Treasury balance:", balanceCheckTreasury2.Tokens, "units of
token", tokenId)

//CHECK THE BALANCE AFTER THE TRANSFER FOR ALICE'S ACCOUNT
balanceCheckAlice2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
fmt.Println("Alice's balance:", balanceCheckAlice2.Tokens, "units of
token", tokenId)
}

```

</details>

```
{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}
```

create-and-transfer-your-first-nft.md:

Create and Transfer Your First NFT

Summary

Using the Hedera Token Service, you can create non-fungible tokens (NFTs). NFTs are uniquely identifiable. On the Hedera network, the token ID represents a collection of NFTs of the same class, and the serial number of each token uniquely identifies each NFT in the class.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment [here](#).

1. Create a Non-Fungible Token (NFT)

Use `TokenCreateTransaction()` to configure and set the token properties. At a minimum, this constructor requires setting a name, symbol, and treasury account ID. All other fields are optional, so if they're not specified then default values are used. For instance, not specifying an admin key, makes a token immutable (can't change or add properties); not specifying a supply key, makes a token supply fixed (can't mint new or burn existing tokens); not specifying a token type, makes a token fungible.

After submitting the transaction to the Hedera network, you can obtain the new token ID by requesting the receipt. This token ID represents an NFT class.

```
{% tabs %}
{% tab title="Java" %}
java
//Create the NFT
TokenCreateTransaction nftCreate = new TokenCreateTransaction()
    .setTokenName("diploma")
    .setTokenSymbol("GRAD")
    .setTokenType(TokenType.NONFUNGIBLEUNIQUE)
    .setDecimals(0)
    .setInitialSupply(0)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.FINITE)
    .setMaxSupply(250)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

//Sign the transaction with the treasury key
TokenCreateTransaction nftCreateTxSign = nftCreate.sign(treasuryKey);

//Submit the transaction to a Hedera network
TransactionResponse nftCreateSubmit = nftCreateTxSign.execute(client);
```



```

//Get the transaction receipt
TransactionReceipt nftCreateRx = nftCreateSubmit.getReceipt(client);

//Get the token ID
TokenId tokenId = nftCreateRx.tokenId;

//Log the token ID
System.out.println("Created NFT with token ID " + tokenId);

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the NFT
const nftCreate = await new TokenCreateTransaction()
    .setTokenName("diploma")
    .setTokenSymbol("GRAD")
    .setTokenType(TokenType.NonFungibleUnique)
    .setDecimals(0)
    .setInitialSupply(0)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.Finite)
    .setMaxSupply(250)
    .setSupplyKey(supplyKey)
    .freezeWith(client);

//Sign the transaction with the treasury key
const nftCreateTxSign = await nftCreate.sign(treasuryKey);

//Submit the transaction to a Hedera network
const nftCreateSubmit = await nftCreateTxSign.execute(client);

//Get the transaction receipt
const nftCreateRx = await nftCreateSubmit.getReceipt(client);

//Get the token ID
const tokenId = nftCreateRx.tokenId;

//Log the token ID
console.log("Created NFT with Token ID: " + tokenId);

{% endtab %}

{% tab title="Go" %}
go
//Create the NFT
nftCreate, err := hedera.NewTokenCreateTransaction().
    SetTokenName("diploma").
    SetTokenSymbol("GRAD").
    SetTokenType(hedera.TokenTypeNonFungibleUnique).
    SetDecimals(0).
    SetInitialSupply(0).
    SetTreasuryAccountID(treasuryAccountId).
    SetSupplyType(hedera.TokenSupplyTypeFinite).
    SetMaxSupply(250).
    SetSupplyKey(supplyKey).
    FreezeWith(client)

//Sign the transaction with the treasury key
nftCreateTxSign := nftCreate.Sign(treasuryKey)

//Submit the transaction to a Hedera network
nftCreateSubmit, err := nftCreateTxSign.Execute(client)

```

```
//Get the transaction receipt
nftCreateRx, err := nftCreateSubmit.GetReceipt(client)

//Get the token ID
tokenId := nftCreateRx.TokenID

//Log the token ID
fmt.Println("Created NFT with token ID ", tokenId)

{% endtab %}
{% endtabs %}
```

2. Mint a New NFT

When creating an NFT, the decimals and initial supply must be set to zero. After the token is created, you mint each NFT using the token mint operation. Specifying a supply key during token creation is a requirement to be able to mint and burn tokens. The supply key is required to sign mint and burn transactions.

Both the NFT image and metadata live in the InterPlanetary File System (IPFS), which provides decentralized storage. The file diploma\metadata.json contains the metadata for the NFT. A content identifier (CID) pointing to the metadata file is used during minting of the new NFT. Notice that the metadata file contains a URI pointing to the NFT image.

```
{% tabs %}
{% tab title="Java" %}
```

```
java
// Max transaction fee as a constant
final int MAXTRANSACTIONFEE = 20;

// IPFS content identifiers for which we will create a NFT
String[] CID = {
    "ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json",
    "ipfs://bafyreic463uarchq4mlufp7pvfkfut7zeqsqmn3b2x3jjxwcjx6b5pk7q/
metadata.json",
    "ipfs://bafyreihhja55q6h2rijscl3gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json",
    "ipfs://bafyreidb23oehkttjbff3gdi4vz7mjiycxjyxadwg32pngod4huozcwphu/
metadata.json",
    "ipfs://bafyreie7ftl6erd5etz5gscfwfiwmht3b52cevdrrf7hjwx5ddns7zneu/
metadata.json"
};

// Mint a new NFT
TokenMintTransaction mintTx = new TokenMintTransaction().setTokenId(tokenId)
    .setMaxTransactionFee(new Hbar(MAXTRANSACTIONFEE));

for (String cid : CID) {
    mintTx.addMetadata(cid.getBytes());
}

// Freeze the transaction
mintTx.freezeWith(client);

// Sign transaction with the supply key
TokenMintTransaction mintTxSign = mintTx.sign(supplyKey);
```

```

// Submit the transaction to a Hedera network
TransactionResponse mintTxSubmit = mintTxSign.execute(client);

// Get the transaction receipt
TransactionReceipt mintRx = mintTxSubmit.getReceipt(client);

// Log the serial number
System.out.println("Created NFT " + tokenId + " with serial: " +
mintRx.serials);

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Max transaction fee as a constant
const maxTransactionFee = new Hbar(20);

//IPFS content identifiers for which we will create a NFT
const CID = [
  Buffer.from(
    "ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreic463uarchq4mlufp7pvfktut7zeqsqmn3b2x3jjxwcjqx6b5pk7q/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreihhja55q6h2rijsc13gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreidb23oehkttjbff3gdi4vz7mjijcxjyxadwg32pngod4huozcwphu/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreie7ftl6erd5etz5gscfwfiwjmht3b52cevdrrf7hjwxx5ddns7zneu/
metadata.json"
  )
];

// MINT NEW BATCH OF NFTs
const mintTx = new TokenMintTransaction()
  .setTokenId(tokenId)
  .setMetadata(CID) //Batch minting - UP TO 10 NFTs in single tx
  .setMaxTransactionFee(maxTransactionFee)
  .freezeWith(client);

//Sign the transaction with the supply key
const mintTxSign = await mintTx.sign(supplyKey);

//Submit the transaction to a Hedera network
const mintTxSubmit = await mintTxSign.execute(client);

//Get the transaction receipt
const mintRx = await mintTxSubmit.getReceipt(client);

//Log the serial number
console.log("Created NFT " + tokenId + " with serial number: " +
mintRx.serials);

{% endtab %}

{% tab title="Go" %}

```

```

go
// Max transaction fee as a constant
const maxTransactionFee = 20 // in tinybars

//IPFS content identifiers for which we will create a NFT
CID := [][]byte{
    []byte("ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json"),
    []byte("ipfs://bafyreic463uarchq4mlufp7pvfkfut7zeqsqmn3b2x3jjxwcj6b5pk7q/
metadata.json"),
    []byte("ipfs://bafyreihhja55q6h2rijscl3gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json"),
    []byte("ipfs://bafyreidb23oehkttjbff3gdi4vz7mjijcxjyxadwg32pngod4huozcwphu/
metadata.json"),
    []byte("ipfs://bafyreie7ftl6erd5etz5gscfwfiwjmh3b52cevd7f7hjwx5ddns7zneu/
metadata.json"),
}

// Mint new NFT
mintTx, err := hedera.NewTokenMintTransaction().
    SetTokenID(tokenId).
    SetMetadata(CID).
    SetMaxTransactionFee(hedera.HbarFromTinybars(maxTransactionFee)).
    FreezeWith(client)

// Sign the transaction with the supply key
mintTxSign := mintTx.Sign(supplyKey)

// Submit the transaction to a Hedera network
mintTxSubmit, err := mintTxSign.Execute(client)

// Get the transaction receipt
mintRx, err := mintTxSubmit.GetReceipt(client)

// Log the serial number
fmt.Printf("Created NFT %s with serial: %d\n", tokenId, mintRx.SerialNumbers[0])

{% endtab %}
{% endtabs %}

{% code title="diplomametadata.json" %}
json
{
    "name": "Diploma",
    "description": "Certificate of Graduation",
    "image":
"https://ipfs.io/ipfs/QmdTNJDYEpD4EUUzYPuhc1GsDELjab6ypgvDQAp4visoM9?
filename=diploma.jpg",
    "properties": {
        "university": "H State University",
        "college": "Engineering and Applied Sciences",
        "level": "Masters",
        "field": "Mechanical Engineering",
        "honors": "yes",
        "honorsType": "Summa Cum Laude",
        "gpa": "3.84",
        "student": "Alice",
        "date": "2021-03-18"
    }
}

{% endcode %}

🔇 Throttle cap warning&#x20;
```

Batch minting of 10+ NFTs can run into throughput issues and potentially hit the transaction limit (throttle cap), causing the SDK to throw BUSY exceptions. Adding an application-level retry loop can help manage these exceptions for the batch-minting process.

The following are examples of retry loops:

```
{% tabs %}
{% tab title="Java" %}
java
private static final int MAXRETRIES = 5;

private static TransactionReceipt executeTransaction
(Transaction<?> transaction, PrivateKey key)
throws Exception {
    int retries = 0;

    while (retries < MAXRETRIES) {
        try {
            TransactionResponse txResponse =
transaction.sign(key).execute(client);
            TransactionReceipt txReceipt = txResponse.getReceipt(client);

            return txReceipt;
        } catch (PrecheckStatusException e) {
            if (e.status == Status.BUSY) {
                retries++;
                System.out.println("Retry attempt: " + retries);
            } else {
                throw e;
            }
        }
    }

    throw new Exception("Transaction failed after " + MAXRETRIES + " attempts");
}

{% endtab %}

{% tab title="JavaScript" %}
javascript
async function executeTransaction(transaction, key) {
    let retries = 0;
    while (retries < MAXRETRIES) {
        try {
            const txSign = await transaction.sign(key);
            const txSubmit = await txSign.execute(client);
            const txReceipt = await txSubmit.getReceipt(client);

            // If the transaction succeeded, return the receipt
            return txReceipt;
        } catch (err) {
            // If the error is BUSY, retry the transaction
            if (err.toString().includes('BUSY')) {
                retries++;
                console.log(Retry attempt: ${retries});
            } else {
                // If the error is not BUSY, throw the error
                throw err;
            }
        }
    }

    throw new Error(Transaction failed after ${MAXRETRIES} attempts);
}
```

```

}

{% endtab %}

{% tab title="Go" %}
go
func retry(attempts int, sleep time.Duration, fn func() error) error {
    err := fn()
    if err != nil {
        if attempts-- > 0 {
            // Exponential backoff
            time.Sleep(sleep)
            return retry(attempts, 2*sleep, fn)
        }
    }
    return err
}

func executeWithRetry(fn func() error) error {
    return retry(2, 1*time.Second, fn)
}

{% endtab %}
{% endtabs %}

```

3. Associate User Accounts with the NFT


Before an account that is not the treasury for a token can receive or send this specific token ID, the account must become “associated” with the token. To associate a token to an account, the account owner must sign the associate transaction. If you have an account with the automatic token association property set, you do not need to associate the token before transferring it to the receiving account.

```

{% tabs %}
{% tab title="Java" %}
java
//Create the associate transaction and sign with Alice's key
TokenAssociateTransaction associateAliceTx = new TokenAssociateTransaction()
    .setAccountId(aliceAccountId)
    .setTokenIds(Collections.singletonList(tokenId))
    .freezeWith(client)
    .sign(aliceKey);

//Submit the transaction to a Hedera network
TransactionResponse associateAliceTxSubmit = associateAliceTx.execute(client);

//Get the transaction receipt
TransactionReceipt associateAliceRx = associateAliceTxSubmit.getReceipt(client);

//Confirm the transaction was successful
System.out.println("\nNFT association with Alice's account: " +
    associateAliceRx.status + " ");

{% endtab %}

{% tab title="JavaScript" %}
javascript
//Create the associate transaction and sign with Alice's key
const associateAliceTx = await new TokenAssociateTransaction()
    .setAccountId(aliceId)
    .setTokenIds([tokenId])

```

```

        .freezeWith(client)
        .sign(aliceKey);

//Submit the transaction to a Hedera network
const associateAliceTxSubmit = await associateAliceTx.execute(client);

//Get the transaction receipt
const associateAliceRx = await associateAliceTxSubmit.getReceipt(client);

//Confirm the transaction was successful
console.log(NFT association with Alice's account: ${associateAliceRx.status}\n);

{% endtab %}

{% tab title="Go" %}
go
//Create the associate transaction
associateAliceTx, err := hedera.NewTokenAssociateTransaction().
    SetAccountID(aliceAccountId).
    SetTokenIDs(tokenId).
    FreezeWith(client)

//Sign with Alice's key
signTx := associateAliceTx.Sign(aliceKey)

//Submit the transaction to a Hedera network
associateAliceTxSubmit, err := signTx.Execute(client)//Get the transaction
receipt

//Get the transaction receipt
associateAliceRx, err := associateAliceTxSubmit.GetReceipt(client)

//Confirm the transaction was successful
fmt.Println("NFT association with Alice's account:", associateAliceRx.Status)

{% endtab %}
{% endtabs %}

```

4. Transfer the NFT

Now, transfer the NFT and check the account balances before and after the send! After the transfer, you should expect 1 NFT to be removed from the treasury account and available in Alice's account. The treasury account key has to sign the transfer transaction to authorize the transfer to Alice's account.

```

{% tabs %}
{% tab title="Java" %}
java
// Check the balance before the NFT transfer for the treasury account
AccountBalance balanceCheckTreasury = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
System.out.println(
    "- Treasury balance: " +
balanceCheckTreasury.tokens.getDefault(tokenId, 0L)
    + " NFTs of ID " + tokenId);

// Check the balance before the NFT transfer for Alice's account
AccountBalance balanceCheckAlice = new
AccountBalanceQuery().setAccountId(aliceAccountId)
    .execute(client);
System.out.println("Alice's balance: " +

```

```

balanceCheckAlice.tokens.getDefault(tokenId, 0L) + " NFTs of ID " + tokenId);

// Transfer NFT from treasury to Alice
// Sign with the treasury key to authorize the transfer
TransferTransaction tokenTransferTx = new TransferTransaction()
    .addNftTransfer(new NftId(tokenId, 1), treasuryId, aliceAccountId)
    .freezeWith(client)
    .sign(treasuryKey);

TransactionResponse tokenTransferSubmit = tokenTransferTx.execute(client);
TransactionReceipt tokenTransferRx = tokenTransferSubmit.getReceipt(client);

System.out.println("\nNFT transfer from Treasury to Alice: " +
    tokenTransferRx.status);

// Check the balance for the treasury account after the transfer
AccountBalance balanceCheckTreasury2 = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
System.out.println("\nTreasury balance: " +
    balanceCheckTreasury2.tokens.getDefault(tokenId, 0L) + " NFTs of ID " +
    tokenId);

// Check the balance for Alice's account after the transfer
AccountBalance balanceCheckAlice2 = new
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
System.out.println(
    "- Alice's balance: " + balanceCheckAlice2.tokens.getDefault(tokenId,
0L)
    + " NFTs of ID " + tokenId + "\n");

{% endtab %}

{% tab title="JavaScript" %}
javascript
// Check the balance before the transfer for the treasury account
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
console.log(Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} NFTs of ID ${tokenId});

// Check the balance before the transfer for Alice's account
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
console.log(Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} NFTs of ID ${tokenId});

// Transfer the NFT from treasury to Alice
// Sign with the treasury key to authorize the transfer
const tokenTransferTx = await new TransferTransaction()
    .addNftTransfer(tokenId, 1, treasuryId, aliceId)
    .freezeWith(client)
    .sign(treasuryKey);

const tokenTransferSubmit = await tokenTransferTx.execute(client);
const tokenTransferRx = await tokenTransferSubmit.getReceipt(client);

console.log(\nNFT transfer from Treasury to Alice: ${tokenTransferRx.status} \
n);

// Check the balance of the treasury account after the transfer
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
console.log(Treasury balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} NFTs of ID ${tokenId});

```



```

// Check the balance of Alice's account after the transfer
var balanceCheckTx = await new
AccountBalanceQuery().setAccountId(aliceId).execute(client);
console.log(Alice's balance: $
{balanceCheckTx.tokens.map.get(tokenId.toString())} NFTs of ID ${tokenId});

{% endtab %}

{% tab title="Go" %}

go
// Transfer the NFT from treasury to Alice
    tokenTransferTx, err := hedera.NewTransferTransaction().
        AddNftTransfer(hedera.NftID{TokenID: tokenId, SerialNumber:
1}, treasuryAccountId, aliceAccountId).
        FreezeWith(client)
    if err != nil {
        panic(err)
    }

    // Sign with the treasury key to authorize the transfer
    signTransferTx := tokenTransferTx.Sign(treasuryKey)

    //Submit the transaction
    tokenTransferSubmit, err := signTransferTx.Execute(client)

    // Get the transaction receipt
    tokenTransferRx, err := tokenTransferSubmit.GetReceipt(client)

    fmt.Println("\nNFT transfer from Treasury to Alice:",
tokenTransferRx.Status, "✅")

    // Check the balance of the treasury account after the transfer
    balanceCheckTreasury2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    treasuryNftBalance2 := balanceCheckTreasury2.Tokens.Get(tokenId)
    fmt.Println("Treasury balance:", treasuryNftBalance2, "NFTs of ID",
tokenId)

    // Check the balance of Alice's account after the transfer
    balanceCheckAlice2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    aliceNftBalance2 := balanceCheckAlice2.Tokens.Get(tokenId)
    fmt.Println("Alice's balance:", aliceNftBalance2, "NFTs of ID",
tokenId, "\n")

{% endtab %}
{% endtabs %}

```

Code Check ☒

<details>

<summary>Java</summary>

```

java
import com.hedera.hashgraph.sdk.;
import io.github.cdimascio.dotenv.Dotenv;

import java.util.;
import java.util.concurrent.TimeoutException;

public class CreateYourFirstNft {

    public static void main(String[] args)
        throws TimeoutException, PrecheckStatusException,
ReceiptStatusException {

        // Load environment variables
        Dotenv dotenv = Dotenv.load();

        // Grab your Hedera testnet account ID and private key
        AccountId myAccountId = AccountId.fromString(dotenv.get("MYACCOUNTID"));
        PrivateKey myPrivateKey =
PrivateKey.fromStringDER(dotenv.get("MYPRIVATEKEY"));

        // Create your Hedera testnet client
        Client client = Client.forTestnet();
        client.setOperator(myAccountId, myPrivateKey);

        // Treasury Key
        PrivateKey treasuryKey = PrivateKey.generateED25519();
        PublicKey treasuryPublicKey = treasuryKey.getPublicKey();

        // Create treasury account
        TransactionResponse treasuryAccount = new AccountCreateTransaction()
            .setKey(treasuryPublicKey)
            .setInitialBalance(new Hbar(10))
            .execute(client);

        AccountId treasuryId = treasuryAccount.getReceipt(client).accountId;

        // Alice Key
        PrivateKey aliceKey = PrivateKey.generateED25519();
        PublicKey alicePublicKey = aliceKey.getPublicKey();

        // Create Alice's account
        TransactionResponse aliceAccount = new AccountCreateTransaction()
            .setKey(alicePublicKey)
            .setInitialBalance(new Hbar(10))
            .execute(client);

        AccountId aliceAccountId = aliceAccount.getReceipt(client).accountId;

        // Generate the supply Key
        PrivateKey supplyKey = PrivateKey.generateED25519();
        PublicKey supplyPublicKey = supplyKey.getPublicKey();

        // Create the NFT
        TokenCreateTransaction nftCreate = new TokenCreateTransaction()
            .setTokenName("diploma")
            .setTokenSymbol("GRAD")
            .setTokenType(TokenType.NONFUNGIBLEUNIQUE)
            .setDecimals(0)
            .setInitialSupply(0)
            .setTreasuryAccountId(treasuryId)
            .setSupplyType(TokenSupplyType.FINITE)
            .setMaxSupply(250)

```

```

        .setSupplyKey(supplyKey)
        .freezeWith(client);

// Sign the transaction with the treasury key
TokenCreateTransaction nftCreateTxSign = nftCreate.sign(treasuryKey);

// Submit the transaction to a Hedera network
TransactionResponse nftCreateSubmit = nftCreateTxSign.execute(client);

// Get the transaction receipt
TransactionReceipt nftCreateRx = nftCreateSubmit.getReceipt(client);

// Get the token ID
TokenId tokenId = nftCreateRx.tokenId;

// Log the token ID
System.out.println("\nCreated NFT with token ID " + tokenId);

// Max transaction fee as a constant
final int MAXTRANSACTIONFEE = 20;

// IPFS content identifiers for which we will create a NFT
String[] CID = {

"ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json",

"ipfs://bafyreic463uarchq4mlufp7pvfkgfut7zeqsqmn3b2x3jjxwcjqx6b5pk7q/
metadata.json",

"ipfs://bafyreihhja55q6h2rijsc13gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json",

"ipfs://bafyreidb23oehkttjbff3gdi4vz7mjiycxjyxadwg32pngod4huozcwphu/
metadata.json",

"ipfs://bafyreie7ftl6erd5etz5gscfwfiwjmht3b52cevdrrf7hjwxx5ddns7zneu/
metadata.json"
};

// Mint a new NFT
TokenMintTransaction mintTx = new
TokenMintTransaction().setTokenId(tokenId)
    .setMaxTransactionFee(new Hbar(MAXTRANSACTIONFEE));

for (String cid : CID) {
    mintTx.addMetadata(cid.getBytes());
}

// Freeze the transaction
mintTx.freezeWith(client);

// Sign transaction with the supply key
TokenMintTransaction mintTxSign = mintTx.sign(supplyKey);

// Submit the transaction to a Hedera network
TransactionResponse mintTxSubmit = mintTxSign.execute(client);

// Get the transaction receipt
TransactionReceipt mintRx = mintTxSubmit.getReceipt(client);

// Log the serial number
System.out.println("Created NFT " + tokenId + " with serial: " +
mintRx.serials);

```

```

        // Create the associate transaction and sign with Alice's key
        TokenAssociateTransaction associateAliceTx = new
TokenAssociateTransaction()
            .setAccountId(aliceAccountId)
            .setTokenIds(Collections.singletonList(tokenId))
            .freezeWith(client)
            .sign(aliceKey);

        // Submit the transaction to a Hedera network
        TransactionResponse associateAliceTxSubmit =
associateAliceTx.execute(client);

        // Get the transaction receipt
        TransactionReceipt associateAliceRx =
associateAliceTxSubmit.getReceipt(client);

        // Confirm the transaction was successful
        System.out.println("\nNFT association with Alice's account: " +
associateAliceRx.status + " ✓");

        // Check the balance before the NFT transfer for the treasury account
        AccountBalance balanceCheckTreasury = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
        System.out.println(
            "- Treasury balance: " +
balanceCheckTreasury.tokens.get(tokenId)
            + " NFTs of ID " + tokenId);

        // Check the balance before the NFT transfer for Alice's account
        AccountBalance balanceCheckAlice = new
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
        System.out.println(
            "- Alice's balance: " + balanceCheckAlice.tokens.get(tokenId)
            + " NFTs of ID " + tokenId);

        // Transfer NFT from treasury to Alice
        // Sign with the treasury key to authorize the transfer
        TransferTransaction tokenTransferTx = new TransferTransaction()
            .addNftTransfer(new NftId(tokenId, 1), treasuryId,
aliceAccountId)
            .freezeWith(client).sign(treasuryKey);

        // Submit the transaction to a Hedera network
        TransactionResponse tokenTransferSubmit =
tokenTransferTx.execute(client);

        // Get the transaction receipt
        TransactionReceipt tokenTransferRx =
tokenTransferSubmit.getReceipt(client);

        // Confirm the transfer was successful
        System.out.println("\nNFT transfer from Treasury to Alice: " +
tokenTransferRx.status + " ✓");

        // Check the balance for the treasury account after the transfer
        AccountBalance balanceCheckTreasury2 = new
AccountBalanceQuery().setAccountId(treasuryId).execute(client);
        System.out.println(
            "- Treasury balance: " +
balanceCheckTreasury2.tokens.get(tokenId)
            + " NFTs of ID " + tokenId);

        // Check the balance for Alice's account after the transfer

```

```

        AccountBalance balanceCheckAlice2 = new
AccountBalanceQuery().setAccountId(aliceAccountId).execute(client);
        System.out.println(
            "- Alice's balance: " + balanceCheckAlice2.tokens.get(tokenId)
                + " NFTs of ID " + tokenId + "\n");
    }
}

```

</details>

<details>

<summary>JavaScript</summary>

```

javascript
console.clear();
require("dotenv").config();
const {
    Hbar,
    Client,
    AccountId,
    PrivateKey,
    TokenType,
    TokenSupplyType,
    TokenMintTransaction,
    TransferTransaction,
    AccountBalanceQuery,
    TokenCreateTransaction,
    TokenAssociateTransaction,
} = require("@hashgraph/sdk");

// Configure accounts and client, and generate needed keys
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromStringDer(process.env.OPERATORPVKEY);
const treasuryId = AccountId.fromString(process.env.TREASURYID);
const treasuryKey = PrivateKey.fromStringDer(process.env.TREASURYPVKEY);
const aliceId = AccountId.fromString(process.env.ALICEID);
const aliceKey = PrivateKey.fromStringDer(process.env.ALICEPVKEY);

const client = Client.forTestnet().setOperator(operatorId, operatorKey);

const supplyKey = PrivateKey.generate();

async function createFirstNft() {
    //Create the NFT
    const nftCreate = await new TokenCreateTransaction()
        .setTokenName("diploma")
        .setTokenSymbol("GRAD")
        .setTokenType(TokenType.NonFungibleUnique)
        .setDecimals(0)
        .setInitialSupply(0)
        .setTreasuryAccountId(treasuryId)
        .setSupplyType(TokenSupplyType.Finite)
        .setMaxSupply(250)
        .setSupplyKey(supplyKey)
        .freezeWith(client);

    //Sign the transaction with the treasury key
    const nftCreateTxSign = await nftCreate.sign(treasuryKey);

    //Submit the transaction to a Hedera network
    const nftCreateSubmit = await nftCreateTxSign.execute(client);
}

```

```

//Get the transaction receipt
const nftCreateRx = await nftCreateSubmit.getReceipt(client);

//Get the token ID
const tokenId = nftCreateRx.tokenId;

//Log the token ID
console.log(`\nCreated NFT with Token ID:  + tokenId`);

// Max transaction fee as a constant
const maxTransactionFee = new Hbar(20);

//IPFS content identifiers for which we will create a NFT
const CID = [
  Buffer.from(
    "ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreic463uarchq4mlufp7pvfkgfut7zeqsqmn3b2x3jjxwcjqx6b5pk7q/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreihhja55q6h2rijsc13gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreidb23oehkttjbff3gdi4vz7mjiycxjyxadwg32pngod4huozcwphu/
metadata.json"
  ),
  Buffer.from(
    "ipfs://bafyreie7ftl6erd5etz5gscfwfiwjmht3b52cevdrrf7hjwxx5ddns7zneu/
metadata.json"
  ),
];

// MINT NEW BATCH OF NFTs
const mintTx = new TokenMintTransaction()
  .setTokenId(tokenId)
  .setMetadata(CID) //Batch minting - UP TO 10 NFTs in single tx
  .setMaxTransactionFee(maxTransactionFee)
  .freezeWith(client);

//Sign the transaction with the supply key
const mintTxSign = await mintTx.sign(supplyKey);

//Submit the transaction to a Hedera network
const mintTxSubmit = await mintTxSign.execute(client);

//Get the transaction receipt
const mintRx = await mintTxSubmit.getReceipt(client);

//Log the serial number
console.log(
  "Created NFT " + tokenId + " with serial number: " + mintRx.serials + "\n"
);

//Create the associate transaction and sign with Alice's key
const associateAliceTx = await new TokenAssociateTransaction()
  .setAccountId(aliceId)
  .setTokenIds([tokenId])
  .freezeWith(client)
  .sign(aliceKey);

```

```

//Submit the transaction to a Hedera network
const associateAliceTxSubmit = await associateAliceTx.execute(client);

//Get the transaction receipt
const associateAliceRx = await associateAliceTxSubmit.getReceipt(client);

//Confirm the transaction was successful
console.log(
  NFT association with Alice's account: ${associateAliceRx.status}\n
);

// Check the balance before the transfer for the treasury account
var balanceCheckTx = await new AccountBalanceQuery()
  .setAccountId(treasuryId)
  .execute(client);
console.log(
  Treasury balance: ${balanceCheckTx.tokens.map.get(
    tokenId.toString()
  )} NFTs of ID ${tokenId}
);

// Check the balance before the transfer for Alice's account
var balanceCheckTx = await new AccountBalanceQuery()
  .setAccountId(aliceId)
  .execute(client);
console.log(
  Alice's balance: ${balanceCheckTx.tokens.map.get(
    tokenId.toString()
  )} NFTs of ID ${tokenId}
);

// Transfer the NFT from treasury to Alice
// Sign with the treasury key to authorize the transfer
const tokenTransferTx = await new TransferTransaction()
  .addNftTransfer(tokenId, 1, treasuryId, aliceId)
  .freezeWith(client)
  .sign(treasuryKey);

const tokenTransferSubmit = await tokenTransferTx.execute(client);
const tokenTransferRx = await tokenTransferSubmit.getReceipt(client);

console.log(
  \nNFT transfer from Treasury to Alice: ${tokenTransferRx.status} \n
);

// Check the balance of the treasury account after the transfer
var balanceCheckTx = await new AccountBalanceQuery()
  .setAccountId(treasuryId)
  .execute(client);
console.log(
  Treasury balance: ${balanceCheckTx.tokens.map.get(
    tokenId.toString()
  )} NFTs of ID ${tokenId}
);

// Check the balance of Alice's account after the transfer
var balanceCheckTx = await new AccountBalanceQuery()
  .setAccountId(aliceId)
  .execute(client);
console.log(
  Alice's balance: ${balanceCheckTx.tokens.map.get(
    tokenId.toString()
  )} NFTs of ID ${tokenId}
);

```

```
}  
createFirstNft();
```

```
</details>
```

```
<details>
```

```
<summary>Go</summary>
```

```
go  
package main  
  
import (  
    "fmt"  
    "os"  
  
    "github.com/hashgraph/hedera-sdk-go/v2"  
    "github.com/joho/godotenv"  
)  
  
func main() {  
    // Load environment variables  
    err := godotenv.Load(".env")  
    if err != nil {  
        panic(fmt.Errorf("Unable to load environment variables from .env  
file. Error:\n%\v\n", err))  
    }  
  
    // Grab your testnet account ID and private key from the .env file  
    myAccountId, err := hedera.AccountIDFromString(os.Getenv("MYACCOUNTID"))  
    if err != nil {  
        panic(err)  
    }  
  
    myPrivateKey, err :=  
hedera.PrivateKeyFromString(os.Getenv("MYPRIVATEKEY"))  
    if err != nil {  
        panic(err)  
    }  
  
    // Create your testnet client  
    client := hedera.ClientForTestnet()  
    client.SetOperator(myAccountId, myPrivateKey)  
  
    // Create a treasury Key  
    treasuryKey, err := hedera.GeneratePrivateKey()  
    if err != nil {  
        panic(err)  
    }  
    treasuryPublicKey := treasuryKey.PublicKey()  
  
    // Create treasury account  
    treasuryAccount, err := hedera.NewAccountCreateTransaction().  
        SetKey(treasuryPublicKey).  
        SetInitialBalance(hedera.NewHbar(10)).  
        Execute(client)  
    if err != nil {  
        panic(err)  
    }  
  
    // Get the receipt of the transaction  
    receipt, err := treasuryAccount.GetReceipt(client)  
    if err != nil {
```



```

        panic(err)
    }

    // Get the account ID
    treasuryAccountId := receipt.AccountID

    // Alice Key
    aliceKey, err := hedera.GeneratePrivateKey()
    alicePublicKey := aliceKey.PublicKey()

    // Create Alice's account
    aliceAccount, err := hedera.NewAccountCreateTransaction().
        SetKey(alicePublicKey).
        SetInitialBalance(hedera.NewHbar(10)).
        Execute(client)

    // Get the receipt of the transaction
    receipt2, err := aliceAccount.GetReceipt(client)

    // Get the account ID
    aliceAccountId := receipt2.AccountID

    // Create a supply key
    supplyKey, err := hedera.GeneratePrivateKey()

    // Create the NFT
    nftCreate, err := hedera.NewTokenCreateTransaction().
        SetTokenName("diploma").
        SetTokenSymbol("GRAD").
        SetTokenType(hedera.TokenTypeNonFungibleUnique).
        SetDecimals(0).
        SetInitialSupply(0).
        SetTreasuryAccountID(treasuryAccountId).
        SetSupplyType(hedera.TokenSupplyTypeFinite).
        SetMaxSupply(250).
        SetSupplyKey(supplyKey).
        FreezeWith(client)

    // Sign the transaction with the treasury key
    nftCreateTxSign := nftCreate.Sign(treasuryKey)

    // Submit the transaction to a Hedera network
    nftCreateSubmit, err := nftCreateTxSign.Execute(client)

    // Get the transaction receipt
    nftCreateRx, err := nftCreateSubmit.GetReceipt(client)

    // Get the token ID
    tokenId := nftCreateRx.TokenID

    // Log the token ID
    fmt.Println("\nCreated NFT with token ID", tokenId)

    // Max transaction fee as a constant
    const maxTransactionFee = 20 // in tinybars

    // IPFS content identifiers for which we will create a NFT
    CID := [][]byte{

[]byte("ipfs://bafyreiao6ajgsfji6qsgbqwdtjdu5gmul7tv2v3pd6kjgcw5o65b2ogst4/
metadata.json"),

```

```
[[]byte("ipfs://bafyreic463uarchq4mlufp7pvfkfut7zeqsqmn3b2x3jjxwcjqx6b5pk7q/
metadata.json"),
```

```
[[]byte("ipfs://bafyreihhja55q6h2rijscl3gra7a3ntiroyglz45z5wlyxdzs6kjh2dinu/
metadata.json"),
```

```
[[]byte("ipfs://bafyreidb23oehkttjbff3gdi4vz7mjijcxjyxadwg32pngod4huozcwphu/
metadata.json"),
```

```
[[]byte("ipfs://bafyreie7ftl6erd5etz5gscfwfiwjmh3b52cevdrrf7hjwx5ddns7zneu/
metadata.json"),
}
```

```
for , singleCID := range CID {
    mintTx, err := hedera.NewTokenMintTransaction().
        SetTokenID(tokenId).
        SetMetadata(singleCID).
        SetMaxTransactionFee(hedera.NewHbar(maxTransactionFee)).
        FreezeWith(client)
    if err != nil {
        fmt.Println("Error while creating mint transaction:", err)
        continue
    }

    mintTxSign := mintTx.Sign(supplyKey)
    mintTxSubmit, err := mintTxSign.Execute(client)
    if err != nil {
        fmt.Println("Error while executing mint transaction:", err)
        continue
    }

    mintRx, err := mintTxSubmit.GetReceipt(client)
    if err != nil {
        fmt.Println("Error while getting mint transaction receipt:",
err)
        continue
    }

    fmt.Printf("Created NFT %s with serial: %d\n", tokenId,
mintRx.SerialNumbers[0])

    // Create the associate transaction
    associateAliceTx, err := hedera.NewTokenAssociateTransaction().
        SetAccountID(aliceAccountId).
        SetTokenIDs(tokenId).
        FreezeWith(client)
    if err != nil {
        panic(err)
    }

    // Sign with Alice's key
    signTx := associateAliceTx.Sign(aliceKey)

    // Submit the transaction to a Hedera network
    associateAliceTxSubmit, err := signTx.Execute(client)

    // Get the transaction receipt
    associateAliceRx, err := associateAliceTxSubmit.GetReceipt(client)

    // Confirm the transaction was successful
    fmt.Println("\nNFT association with Alice's account:",
```

```

associateAliceRx.Status, "✅")

    // Check the balance before the transfer for the treasury account
    balanceCheckTreasury, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    treasuryNftBalance := balanceCheckTreasury.Tokens.Get(tokenId)
    fmt.Println("- Treasury balance:", treasuryNftBalance, "NFTs of ID",
tokenId)

    // Check the balance before the transfer for Alice's account
    balanceCheckAlice, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    aliceNftBalance := balanceCheckAlice.Tokens.Get(tokenId)
    fmt.Println("- Alice's balance:", aliceNftBalance, "NFTs of ID",
tokenId)

    // Transfer the NFT from treasury to Alice
    tokenTransferTx, err := hedera.NewTransferTransaction().
        AddNftTransfer(hedera.NftID{TokenID: tokenId, SerialNumber:
1}, treasuryAccountId, aliceAccountId).
        FreezeWith(client)
    if err != nil {
        panic(err)
    }

    // Sign with the treasury key to authorize the transfer
    signTransferTx := tokenTransferTx.Sign(treasuryKey)

    tokenTransferSubmit, err := signTransferTx.Execute(client)
    tokenTransferRx, err := tokenTransferSubmit.GetReceipt(client)

    fmt.Println("\nNFT transfer from Treasury to Alice:",
tokenTransferRx.Status, "✅")

    // Check the balance of the treasury account after the transfer
    balanceCheckTreasury2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(treasuryAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    treasuryNftBalance2 := balanceCheckTreasury2.Tokens.Get(tokenId)
    fmt.Println("- Treasury balance:", treasuryNftBalance2, "NFTs of ID
", tokenId)

    // Check the balance of Alice's account after the transfer
    balanceCheckAlice2, err :=
hedera.NewAccountBalanceQuery().SetAccountID(aliceAccountId).Execute(client)
    if err != nil {
        panic(err)
    }
    aliceNftBalance2 := balanceCheckAlice2.Tokens.Get(tokenId)
    fmt.Println("- Alice's balance:", aliceNftBalance2, "NFTs of ID ",
tokenId, "\n")
    }
}

```

</details>

{% hint style="info" %}
Have a question? Ask it on StackOverflow
{% endhint %}

hedera-token-service-part-1-how-to-mint-nfts.md:

Hedera Token Service - Part 1: How to Mint NFTs

Hedera Token Service (HTS) enables you to configure, mint, and manage tokens on the Hedera network without the need to set up and deploy a smart contract. Tokens are as fast, fair, and secure as HBAR and cost a fraction of 1¢ USD to transfer.

{% embed url="https://youtu.be/lp3mwdYEZEK?si=DN-IMn03bz8eG6u3" %}

Let's look at some of the functionality available to you with HTS. You will see that the ease of use and flexibility this service provides make HTS tokens an excellent alternative to tokens with smart contracts on Ethereum. For those coming from Ethereum, HTS functionality can be mapped to multiple types of ERC token standards, including ERC20, ERC721, and ERC1155 – you can learn more about the mapping in this blog post. Starting in early 2022, you can use HTS with smart contracts for cases needing advanced logic and programmability for your tokens.

In this part of the series, you will learn how to:

- Create a custom fee schedule
- Configure a non-fungible token (NFT)
- Mint and burn NFTs
- Associate and Transfer NFTs

We will configure an NFT art collection for autumn images. With HTS, you can also create fungible tokens representing anything from a stablecoin pegged to the USD value to an in-game reward system.

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

☑ If you want the entire code used for this tutorial, skip to the Code Check section below.

{% hint style="info" %}
Note: While the following examples are in JavaScript, official SDKs supporting Go and Java are also available and implemented very similarly alongside community-supported SDKs in .NET and various other frameworks or languages.
{% endhint %}

Create New Hedera Accounts and Generate Keys for the NFT

Let's create additional Hedera accounts to represent users for this scenario, such as the Treasury, Alice, and Bob. These accounts are created using your Testnet account credentials from the Hedera portal (see the resources for getting started). Account creation starts by generating a private key for the new account and then calling a reusable function (accountCreatorFcn) that uses the new key, an initial balance, and the Hedera client. You can easily reuse

this function if you need to create more accounts in the future.

Once accounts are created for Treasury, Alice, and Bob, new private keys are generated to manage specific token functionality. Always provide the corresponding public key when specifying the key value for specific functionality.

```
{% hint style="warning" %}
Never expose or share your private key(s) with others, as that may result in
lost funds or loss of control over your account.
{% endhint %}

{% code title="nft-part1.js" %}
javascript
// CREATE NEW HEDERA ACCOUNTS TO REPRESENT OTHER USERS
const initBalance = new Hbar(200);
const treasuryKey = PrivateKey.generateECDSA();
const [treasurySt, treasuryId] = await accountCreateFcn(
  treasuryKey,
  initBalance,
  client
);
console.log(- Treasury's account: https://hashscan.io/testnet/account/${
  treasuryId});
const aliceKey = PrivateKey.generateECDSA();
const [aliceSt, aliceId] = await accountCreateFcn(aliceKey, initBalance,
  client);
console.log(- Alice's account: https://hashscan.io/testnet/account/${aliceId});
const bobKey = PrivateKey.generateECDSA();
const [bobSt, bobId] = await accountCreateFcn(bobKey, initBalance, client);
console.log(- Bob's account: https://hashscan.io/testnet/account/${bobId});

// GENERATE KEYS TO MANAGE FUNCTIONAL ASPECTS OF THE TOKEN
const supplyKey = PrivateKey.generate();
const adminKey = PrivateKey.generate();
const pauseKey = PrivateKey.generate();
const freezeKey = PrivateKey.generate();
const wipeKey = PrivateKey.generate();
const kycKey = PrivateKey.generate();
const newKycKey = PrivateKey.generate();

{% endcode %}

{% code title="nft-part1.js" %}
javascript
// ACCOUNT CREATOR FUNCTION =====
async function accountCreateFcn(pvKey, iBal, client) {
  const response = await new AccountCreateTransaction()
    .setInitialBalance(iBal)
    .setKey(pvKey.publicKey)
    .setMaxAutomaticTokenAssociations(10)
    .execute(client);
  const receipt = await response.getReceipt(client);
  return [receipt.status, receipt.accountId];
}

{% endcode %}
```

Console output:

```
bash
- Treasury's account: https://hashscan.io/testnet/account/0.0.4672116
- Alice's account: https://hashscan.io/testnet/account/0.0.4672117
- Bob's account: https://hashscan.io/testnet/account/0.0.4672118
```

Create a Custom Fee Schedule

Let's start by defining the custom fees for the NFT. Custom fees are distributed to the specified accounts each time the token is transferred. Depending on the token type (fungible or non-fungible), you can specify a custom fee to be fixed, fractional, or a royalty. An NFT can only have fixed or royalty fees, so in this example, we'll go with a royalty fee. This enables collecting a fraction of the value exchanged for the NFT when ownership is transferred from one person to another.

```
{% code title="nft-part1.js" %}
javascript
// DEFINE CUSTOM FEE SCHEDULE
let nftCustomFee = await new CustomRoyaltyFee()
    .setNumerator(1)
    .setDenominator(10)
    .setFeeCollectorAccountId(treasuryId)
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(200)));

{% endcode %}
```

Create a Non-Fungible Token (NFT)

These are the images for our NFT collection.

```
<figure><figcaption></figcaption></figure>
```

The images and their metadata live in the InterPlanetary File System (IPFS), which provides decentralized storage. In the next section, we will need the metadata to mint NFTs. For the metadata, we use the standard in this specification.

These are the content identifiers (CIDs) for the NFT metadata, which points to the images, and below is a sample of the metadata:

```
{% code title="nft-part1.js" %}
javascript
// IPFS CONTENT IDENTIFIERS FOR WHICH WE WILL CREATE NFTs - SEE
uploadJsonToIpfs.js
let CIDs = [

Buffer.from("ipfs://bafkreibr7cyxmy4iyckmlyzige4ywccyygomwrcn4ldcldacw3nxe3ikgq"
),

Buffer.from("ipfs://bafkreig73xgqp7wy7qvjwz33rp3nqxaxqlsb7v3id24poe2dath7pj5dhe"
),

Buffer.from("ipfs://bafkreiglqt4oaofxll3o2cc3e3q3ofqzu6puennmambpulxexo5sryc6e"
),

Buffer.from("ipfs://bafkreiaoswszev3uoukkepctzpnzw56ey6w3xscokvsvmfrrqdzmyhas6fu"
),

Buffer.from("ipfs://bafkreih6cajqynaqwbrmiabk2jxpy56rpf25zvg5lbien73p5ysnpehyjm"
),
];

{% endcode %}

javascript
{
```

```

"name": "LEAF1",
"creator": "Mother Nature & Hashgraph",
"description": "Autumn",
"image": "ipfs://Qmb3CMWJzxWZJ34TgJgjASvdTc4x6PEz6LGm2QTWPPpkw5",
"type": "image/jpg",
"format": "HIP412@2.0.0",
"properties": {
  "city": "Boston",
  "season": "Fall",
  "decade": "20's",
  "license": "MIT-0",
  "collection": "Fall Collection",
  "website": "www.hashgraph.com"
}
}

```

Now, let's create the token. Use `TokenCreateTransaction()` to configure and set the token properties. At a minimum, this constructor requires setting a name, symbol, and treasury account ID. All other fields are optional, so default values are used if they're not specified. For instance, not specifying an admin key, makes a token immutable (can't change or add properties); not specifying a supply key, makes a token supply fixed (can't mint new or burn existing tokens); not specifying a token type, makes a token fungible; for more info on the defaults check out the documentation.

After submitting the transaction to the Hedera network, you can obtain the new token ID by requesting the receipt. This token ID represents an NFT class.

```

{% code title="nft-part1.js" %}
javascript
// CREATE NFT WITH CUSTOM FEE
let nftCreateTx = await new TokenCreateTransaction()
  .setTokenName("Fall Collection")
  .setTokenSymbol("LEAF")
  .setTokenType(TokenType.NonFungibleUnique)
  .setDecimals(0)
  .setInitialSupply(0)
  .setTreasuryAccountId(treasuryId)
  .setSupplyType(TokenSupplyType.Finite)
  .setMaxSupply(CIDs.length)
  .setCustomFees([nftCustomFee])
  .setAdminKey(adminKey)
  .setSupplyKey(supplyKey)
  .setPauseKey(pauseKey)
  .setFreezeKey(freezeKey)
  .setWipeKey(wipeKey)
  .freezeWith(client)
  .sign(treasuryKey);

let nftCreateTxSign = await nftCreateTx.sign(adminKey)
let nftCreateSubmit = await nftCreateTxSign.execute(client);
let nftCreateRx = await nftCreateSubmit.getReceipt(client);
let tokenId = nftCreateRx.tokenId;
console.log(`\n- Created NFT with Token ID: ${tokenId}`);
console.log(
  - See: https://hashscan.io/\${network}/transaction/\${nftCreateSubmit.transactionId}
);

// TOKEN QUERY TO CHECK THAT THE CUSTOM FEE SCHEDULE IS ASSOCIATED WITH NFT
var tokenInfo = await new TokenInfoQuery().setTokenId(tokenId).execute(client);
console.log( );
console.table(tokenInfo.customFees[0]);

```

```
{% endcode %}
```

Console output:

```
bash
```

```
- Created NFT with Token ID: 0.0.4672119  
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230932.976832324
```

```
<figure><figcaption></figcaption></figure>
```

Mint and Burn NFTs

In the code above for the NFT creation, the decimals and initial supply must be set to zero. Once the token is created, you will have to mint each NFT using the token mint operation. Specifying a supply key during token creation is a requirement to be able to mint and burn tokens.

In terms of use cases, you may want to mint new NFTs to add items to your NFT class, or you may need to burn NFTs to take a specific item out of circulation. Alternatively, if you're working with a fungible token (like a stablecoin), you may want to mint new tokens every time there is a new deposit and burn tokens anytime that someone converts their tokens back into fiat.

In this case we're creating a batch of five NFTs for a collection of five images. We'll use a "token minter" function and a for loop to speed up the batch NFT minting from our array of content identifiers (CID array):

```
{% code title="nft-part1.js" %}  
javascript  
// MINT NEW BATCH OF NFTs - CAN MINT UP TO 10 NFT SERIALS IN A SINGLE  
TRANSACTION  
let [nftMintRx, mintTxId] = await tokenMinterFcn(CIDs);  
console.log(  
  \n- Mint ${CIDs.length} serials for NFT collection ${tokenId}: $  
{nftMintRx.status}  
);  
console.log(- See: https://hashscan.io/${network}/transaction/${mintTxId});  
  
{% endcode %}
```

```
{% code title="nft-part1.js" %}  
javascript  
// TOKEN MINTER FUNCTION =====  
async function tokenMinterFcn(CIDs) {  
  let mintTx = new TokenMintTransaction()  
    .setTokenId(tokenId)  
    .setMetadata(CIDs)  
    .freezeWith(client);  
  let mintTxSign = await mintTx.sign(supplyKey);  
  let mintTxSubmit = await mintTxSign.execute(client);  
  let mintRx = await mintTxSubmit.getReceipt(client);  
  return [mintRx, mintTxSubmit.transactionId];  
}  
  
{% endcode %}
```

Console output:

```
bash
```

```
- Mint 5 serials for NFT collection 0.0.4672119: SUCCESS  
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230934.771266715
```


If you change your mind and decide that you don't need the last NFT, then you can burn it as follows:

```
{% code title="nft-part1.js" %}
javascript
// BURN THE LAST NFT IN THE COLLECTION
let tokenBurnTx = await new TokenBurnTransaction()
    .setTokenId(tokenId)
    .setSerials([CIDs.length])
    .freezeWith(client)
    .sign(supplyKey);

let tokenBurnSubmit = await tokenBurnTx.execute(client);
let tokenBurnRx = await tokenBurnSubmit.getReceipt(client);
console.log(\n- Burn NFT with serial ${CIDs.length}: ${tokenBurnRx.status});
console.log(
  - See: https://hashscan.io/${network}/transaction/${tokenBurnSubmit.transactionId}
);

var tokenInfo = await new TokenInfoQuery()
    .setTokenId(tokenId)
    .execute(client);
console.log(\n- Current NFT supply: ${tokenInfo.totalSupply});

{% endcode %}
```

Console output:

```
bash
- Burn NFT with serial 5: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230939.588918306
- Current NFT supply: 4
```

Auto-Associate and Transfer NFTs

Before an account that is not the treasury for a token can receive or send this specific token ID, it must become "associated" with the token. This helps reduce unwanted spam and other concerns from users who don't want to be associated with any of the variety of tokens created on the Hedera network.

This association between an account and a token ID can be done in two ways, manually or automatically. Note that automatic associations can be done for both existing and newly created accounts. For the purposes of our example, we'll do both.

Alice's account will be updated to associate with the token automatically
Bob's account will be manually associated with the token ID

```
{% code title="nft-part1.js" %}
javascript
// AUTO-ASSOCIATION FOR ALICE'S ACCOUNT
let associateTx = await new AccountUpdateTransaction()
    .setAccountId(aliceId)
    .setMaxAutomaticTokenAssociations(10)
    .freezeWith(client)
    .sign(aliceKey);
let associateTxSubmit = await associateTx.execute(client);
let associateRx = await associateTxSubmit.getReceipt(client);
console.log(\n- Alice NFT Auto-Association: ${associateRx.status});
console.log(
```

```

- See: https://hashscan.io/${network}/transaction/${
associateTxSubmit.transactionId}
);

// MANUAL ASSOCIATION FOR BOB'S ACCOUNT
let associateBobTx = await new TokenAssociateTransaction()
  .setAccountId(bobId)
  .setTokenIds([tokenId])
  .freezeWith(client)
  .sign(bobKey);
let associateBobTxSubmit = await associateBobTx.execute(client);
let associateBobRx = await associateBobTxSubmit.getReceipt(client);
console.log(\n- Bob NFT Manual Association: ${associateBobRx.status});
console.log(
- See: https://hashscan.io/${network}/transaction/${
associateBobTxSubmit.transactionId}
);

{% encode %}

```

Console output:

```

bash
- Alice NFT auto-association: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230939.742284556

- Bob NFT manual association: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230942.397064432

```

Finally, let's do two transfers of the NFT with serial number 2 and see how the royalty fees are collected. The first transfer will be from the Treasury to Alice, and the second NFT transfer will be from Alice to Bob in exchange for 100 HBAR.

NFT Transfer from Treasury to Alice

Now, let's do the first NFT transfer and check the account balances before and after the send.

```

{% code title="nft-part1.js" %}
javascript
// BALANCE CHECK 1
oB = await bCheckerFcn(treasuryId);
aB = await bCheckerFcn(aliceId);
bB = await bCheckerFcn(bobId);
console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and ${oB[1]});
console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

// 1st TRANSFER NFT Treasury->Alice
let tokenTransferTx = await new TransferTransaction()
  .addNftTransfer(tokenId, 2, treasuryId, aliceId)
  .freezeWith(client)
  .sign(treasuryKey);
let tokenTransferSubmit = await tokenTransferTx.execute(client);
let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);
console.log(\n NFT transfer Treasury->Alice status: ${tokenTransferRx.status});
console.log(
- See: https://hashscan.io/${network}/transaction/${
tokenTransferSubmit.transactionId}
);

// BALANCE CHECK 2: COPY/PASTE THE CODE ABOVE IN BALANCE CHECK 1

```

```
// BALANCE CHECK 2
oB = await bCheckerFcn(treasuryId);
aB = await bCheckerFcn(aliceId);
bB = await bCheckerFcn(bobId);
console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and ${oB[1]});
console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});
```

```
{% endcode %}
```

```
{% code title="nft-part1.js" %}
```

```
javascript
```

```
// BALANCE CHECKER FUNCTION =====
```

```
async function bCheckerFcn(id) {
  balanceCheckTx = await new AccountBalanceQuery()
    .setAccountId(id)
    .execute(client);
  return [
    balanceCheckTx.tokens.map.get(tokenId.toString()),
    balanceCheckTx.hbars,
  ];
}
```

```
{% endcode %}
```

Console output:

```
bash
```

```
- Treasury balance: 4 NFTs of ID:0.0.4672119 and 1 ¢
- Alice balance: undefined NFTs of ID:0.0.4672119 and 1 ¢
- Bob balance: 0 NFTs of ID:0.0.4672119 and 1 ¢

- NFT transfer Treasury -> Alice status: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230943.472094366

- Treasury balance: 3 NFTs of ID:0.0.4672119 and 1 ¢
- Alice balance: 1 NFTs of ID:0.0.4672119 and 1 ¢
- Bob balance: 0 NFTs of ID:0.0.4672119 and 1 ¢
```

As you remember from the Custom Token Fees documentation, the treasury account and any fee-collecting accounts for a token are exempt from paying custom transaction fees when the token is transferred. Since the treasury account is also the fee collector for the token, that means there are no royalty fees collected in this first transfer.

NFT Transfer from Alice to Bob

```
{% code title="nft-part1.js" %}
```

```
javascript
```

```
// 2nd NFT TRANSFER NFT Alice->Bob
```

```
let nftPrice = new Hbar(10000000, HbarUnit.Tinybar); // 1HBAR = 10,000,000 Tinybar
```

```
let tokenTransferTx2 = await new TransferTransaction()
  .addNftTransfer(tokenId, 2, aliceId, bobId)
  .addHbarTransfer(aliceId, nftPrice)
  .addHbarTransfer(bobId, nftPrice.negated())
  .freezeWith(client)
  .sign(aliceKey);
let tokenTransferTx2Sign = await tokenTransferTx2.sign(bobKey);
let tokenTransferSubmit2 = await tokenTransferTx2Sign.execute(client);
let tokenTransferRx2 = await tokenTransferSubmit2.getReceipt(client);
console.log(\n NFT transfer Alice->Bob status: ${tokenTransferRx2.status});
console.log(
```

```
- See: https://hashscan.io/${network}/transaction/${tokenTransferSubmit2.transactionId}
);
```

```
// BALANCE CHECK 3: COPY/PASTE THE CODE ABOVE IN BALANCE CHECK 1
```

```
{% encode %}
```

Console output:

```
bash
```

```
- NFT transfer Alice -> Bob status: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4649505@1723230945.514991505

- Treasury balance: 3 NFTs of ID:0.0.4672119 and 1.01 ¢
- Alice balance: 0 NFTs of ID:0.0.4672119 and 1.09 ¢
- Bob balance: 1 NFTs of ID:0.0.4672119 and 0.9 ¢
```

Remember from the documentation that royalty fees are paid from the fungible value exchanged, which was 100 HBAR in this case. The royalty fee is specified to be 50%, so that's why the treasury collects 50 HBAR, and Alice collects the remaining 50 HBAR. Remember that when there's no exchange of fungible value (like HBAR or a fungible token), the fallback fee is charged (currently 200 HBAR in our custom fee schedule).

Conclusion

You just learned how to create an NFT on the Hedera network at the native layer without the need to code complex smart contracts! You can create, mint, burn, associate, and transfer NFTs with just a few lines of code in your favorite programming language. Continue to Part 2 to learn how to work with compliance features like Know your Customer (KYC), update tokens, and schedule transactions. Then in Part 3, you will see how to pause, freeze, wipe, and delete tokens.

<details>

<summary>Code Check ☒</summary>

```
{% code title="nft-pt1.js" %}
javascript
console.clear();
require("dotenv").config();
```

```
const {
  AccountId,
  PrivateKey,
  Client,
  TokenCreateTransaction,
  TokenInfoQuery,
  TokenType,
  CustomRoyaltyFee,
  CustomFixedFee,
  Hbar,
  HbarUnit,
  TokenSupplyType,
  TokenMintTransaction,
  TokenBurnTransaction,
  TransferTransaction,
  AccountBalanceQuery,
  AccountUpdateTransaction,
```

```

TokenAssociateTransaction,
TokenNftInfoQuery,
NftId,
AccountCreateTransaction,
} = require("@hashgraph/sdk");

// CONFIGURE ACCOUNTS AND CLIENT, AND GENERATE accounts and client, and
generate needed keys
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromStringECDSA(process.env.OPERATORKEYHEX);
const network = process.env.NETWORK;

const client = Client.forNetwork(network).setOperator(operatorId, operatorKey);
client.setDefaultMaxTransactionFee(new Hbar(50));
client.setDefaultMaxQueryPayment(new Hbar(1));

async function main() {
  // CREATE NEW HEDERA ACCOUNTS TO REPRESENT OTHER USERS
  const initBalance = new Hbar(1);

  const treasuryKey = PrivateKey.generateECDSA();
  const [treasurySt, treasuryId] = await accountCreateFcn(
    treasuryKey,
    initBalance,
    client
  );
  console.log(
    - Treasury's account: https://hashscan.io/testnet/account/${treasuryId}
  );
  const aliceKey = PrivateKey.generateECDSA();
  const [aliceSt, aliceId] = await accountCreateFcn(
    aliceKey,
    initBalance,
    client
  );
  console.log(
    - Alice's account: https://hashscan.io/testnet/account/${aliceId}
  );
  const bobKey = PrivateKey.generateECDSA();
  const [bobSt, bobId] = await accountCreateFcn(bobKey, initBalance, client);
  console.log(- Bob's account: https://hashscan.io/testnet/account/${bobId});

  // GENERATE KEYS TO MANAGE FUNCTIONAL ASPECTS OF THE TOKEN
  const supplyKey = PrivateKey.generateECDSA();
  const adminKey = PrivateKey.generateECDSA();
  const pauseKey = PrivateKey.generateECDSA();
  const freezeKey = PrivateKey.generateECDSA();
  const wipeKey = PrivateKey.generateECDSA();
  const kycKey = PrivateKey.generate();
  const newKycKey = PrivateKey.generate();

  // DEFINE CUSTOM FEE SCHEDULE
  let nftCustomFee = new CustomRoyaltyFee()
    .setNumerator(1)
    .setDenominator(10)
    .setFeeCollectorAccountId(treasuryId)
    .setFallbackFee(
      new CustomFixedFee().setHbarAmount(new Hbar(1, HbarUnit.Tinybar))
    ); // 1 HBAR = 100,000,000 Tinybar

  // IPFS CONTENT IDENTIFIERS FOR WHICH WE WILL CREATE NFTs - SEE
  uploadJsonToIpfs.js
  let CIDs = [
    Buffer.from(

```

```

        "ipfs://bafkreibr7cyxmy4iyckmlyzige4ywccyygomwrcn4ldcldacw3nxe3ikgq"
    ),
    Buffer.from(
        "ipfs://bafkreig73xgqp7wy7qvjwz33rp3nqxaxqlsb7v3id24poe2dath7pj5dhe"
    ),
    Buffer.from(
        "ipfs://bafkreigltq4oaofxll3o2cc3e3q3ofqzu6puennmambpulxexo5sryc6e"
    ),
    Buffer.from(
        "ipfs://bafkreiaoswszev3uoukkepctzpnzw56ey6w3xscokvsvmfrqdzmyhas6fu"
    ),
    Buffer.from(
        "ipfs://bafkreih6cajqynaqwbrmiabk2jxpy56rpf25zvg5lbien73p5ysnpehyjm"
    ),
];

// CREATE NFT WITH CUSTOM FEE
let nftCreateTx = await new TokenCreateTransaction()
    .setTokenName("Fall Collection")
    .setTokenSymbol("LEAF")
    .setTokenType(TokenType.NonFungibleUnique)
    .setDecimals(0)
    .setInitialSupply(0)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.Finite)
    .setMaxSupply(CIDs.length)
    .setCustomFees([nftCustomFee])
    .setAdminKey(adminKey.publicKey)
    .setSupplyKey(supplyKey.publicKey)
    .setPauseKey(pauseKey.publicKey)
    .setFreezeKey(freezeKey.publicKey)
    .setWipeKey(wipeKey.publicKey)
    .freezeWith(client)
    .sign(treasuryKey);

let nftCreateTxSign = await nftCreateTx.sign(adminKey);
let nftCreateSubmit = await nftCreateTxSign.execute(client);
let nftCreateRx = await nftCreateSubmit.getReceipt(client);
let tokenId = nftCreateRx.tokenId;
console.log(`\n- Created NFT with Token ID: ${tokenId}`);
console.log(
    - See: https://hashscan.io/${network}/transaction/${
        nftCreateSubmit.transactionId
    }
);

// TOKEN QUERY TO CHECK THAT THE CUSTOM FEE SCHEDULE IS ASSOCIATED WITH NFT
var tokenInfo = await new TokenInfoQuery()
    .setTokenId(tokenId)
    .execute(client);
console.log( );
console.table(tokenInfo.customFees[0]);

// MINT NEW BATCH OF NFTs - CAN MINT UP TO 10 NFT SERIALS IN A SINGLE TRANSACTION
let [nftMintRx, mintTxId] = await tokenMinterFcn(CIDs);
console.log(
    \n- Mint ${CIDs.length} serials for NFT collection ${tokenId}: $
    {nftMintRx.status}
);
console.log(- See: https://hashscan.io/${network}/transaction/${mintTxId});

// BURN THE LAST NFT IN THE COLLECTION
let tokenBurnTx = await new TokenBurnTransaction()
    .setTokenId(tokenId)

```

```

        .setSerials([CIDs.length])
        .freezeWith(client)
        .sign(supplyKey);
let tokenBurnSubmit = await tokenBurnTx.execute(client);
let tokenBurnRx = await tokenBurnSubmit.getReceipt(client);
console.log(\n- Burn NFT with serial ${CIDs.length}: ${tokenBurnRx.status});
console.log(
    - See: https://hashscan.io/${network}/transaction/${tokenBurnSubmit.transactionId}
);

var tokenInfo = await new TokenInfoQuery()
    .setTokenId(tokenId)
    .execute(client);
console.log(- Current NFT supply: ${tokenInfo.totalSupply});

// AUTO-ASSOCIATION FOR ALICE'S ACCOUNT
let associateTx = await new AccountUpdateTransaction()
    .setAccountId(aliceId)
    .setMaxAutomaticTokenAssociations(10)
    .freezeWith(client)
    .sign(aliceKey);
let associateTxSubmit = await associateTx.execute(client);
let associateRx = await associateTxSubmit.getReceipt(client);
console.log(\n- Alice NFT auto-association: ${associateRx.status});
console.log(
    - See: https://hashscan.io/${network}/transaction/${associateTxSubmit.transactionId}
);

// MANUAL ASSOCIATION FOR BOB'S ACCOUNT
let associateBobTx = await new TokenAssociateTransaction()
    .setAccountId(bobId)
    .setTokenIds([tokenId])
    .freezeWith(client)
    .sign(bobKey);
let associateBobTxSubmit = await associateBobTx.execute(client);
let associateBobRx = await associateBobTxSubmit.getReceipt(client);
console.log(\n- Bob NFT manual association: ${associateBobRx.status});
console.log(
    - See: https://hashscan.io/${network}/transaction/${associateBobTxSubmit.transactionId}
);

// BALANCE CHECK 1
oB = await bCheckerFcn(treasuryId);
aB = await bCheckerFcn(aliceId);
bB = await bCheckerFcn(bobId);
console.log(
    \n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and ${oB[1]}
);
console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

// 1st TRANSFER NFT Treasury -> Alice
let tokenTransferTx = await new TransferTransaction()
    .addNftTransfer(tokenId, 2, treasuryId, aliceId)
    .freezeWith(client)
    .sign(treasuryKey);
let tokenTransferSubmit = await tokenTransferTx.execute(client);
let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);
console.log(
    \n- NFT transfer Treasury -> Alice status: ${tokenTransferRx.status}
);

```

```

    console.log(
      - See: https://hashscan.io/\${network}/transaction/\${tokenTransferSubmit.transactionId}
    );

    // BALANCE CHECK 2
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(
      \n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and ${oB[1]}
    );
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    // 2nd NFT TRANSFER NFT Alice -> Bob
    let nftPrice = new Hbar(10000000, HbarUnit.Tinybar); // 1 HBAR = 10,000,000 Tinybar

    let tokenTransferTx2 = await new TransferTransaction()
      .addNftTransfer(tokenId, 2, aliceId, bobId)
      .addHbarTransfer(aliceId, nftPrice)
      .addHbarTransfer(bobId, nftPrice.negated())
      .freezeWith(client)
      .sign(aliceKey);
    let tokenTransferTx2Sign = await tokenTransferTx2.sign(bobKey);
    let tokenTransferSubmit2 = await tokenTransferTx2Sign.execute(client);
    let tokenTransferRx2 = await tokenTransferSubmit2.getReceipt(client);
    console.log(
      \n- NFT transfer Alice -> Bob status: ${tokenTransferRx2.status}
    );
    console.log(
      - See: https://hashscan.io/\${network}/transaction/\${tokenTransferSubmit2.transactionId}
    );

    // BALANCE CHECK 3
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(
      \n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and ${oB[1]}
    );
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    console.log(
      \n- THE END =====\n
    );
    console.log(- 🖱 Go to:);
    console.log(- 🌐 www.hedera.com/discord\n);

    client.close();

    // ACCOUNT CREATOR FUNCTION =====
    async function accountCreateFcn(pvKey, iBal, client) {
      const response = await new AccountCreateTransaction()
        .setInitialBalance(iBal)
        .setKey(pvKey.publicKey)
        .setMaxAutomaticTokenAssociations(10)
        .execute(client);
      const receipt = await response.getReceipt(client);
      return [receipt.status, receipt.accountId];
    }

```



```
// TOKEN MINTER FUNCTION =====
async function tokenMinterFcn(CIDs) {
  let mintTx = new TokenMintTransaction()
    .setTokenId(tokenId)
    .setMetadata(CIDs)
    .freezeWith(client);
  let mintTxSign = await mintTx.sign(supplyKey);
  let mintTxSubmit = await mintTxSign.execute(client);
  let mintRx = await mintTxSubmit.getReceipt(client);
  return [mintRx, mintTxSubmit.transactionId];
}


// BALANCE CHECKER FUNCTION =====
async function bCheckerFcn(id) {
  balanceCheckTx = await new AccountBalanceQuery()
    .setAccountId(id)
    .execute(client);
  return [
    balanceCheckTx.tokens.map.get(tokenId.toString()),
    balanceCheckTx.hbars,
  ];
}
}


main();

{% endcode %}
```

</details>

Additional Resources

 Project Repository

 Have a question? Ask on StackOverflow

<p>align="center"><p>Writer: Ed, DevRel Engineer</p><p>GitHub LinkedIn</p></td><td>https://www.linkedin.com/in/ed-marquez/</td></tr><tr><td align="center"><p>Editor: Krystal, Technical Writer</p><p>GitHub Twitter</p></td><td>https://x.com/theekrystallee</td></tr></tbody></table></p>

hedera-token-service-part-2-kyc-update-and-scheduled-transactions.md:

Hedera Token Service - Part 2: KYC, Update, and Scheduled Transactions

In Part 1 of the series, you saw how to mint and transfer an NFT using the Hedera Token Service (HTS). Now, in Part 2, you will see how to take advantage of the flexibility you get when you create and configure your tokens with HTS. More specifically, you will learn how to:

- Enable and disable a KYC flag for a token (KYC stands for “know your customer”)
- Update token properties (only possible if it’s a mutable token. Hint: AdminKey)
- Schedule transactions (like a token transfer)

```
{% embed url="https://youtu.be/sxs530LnF48" %}
```

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

Get a Hedera testnet account.
Set up your environment here.

☑ If you want the entire code used for this tutorial, skip to the Code Check section below.

Understanding KYC and Hedera Tokens

KYC Key

When you create a token with HTS, you can optionally use the `.setKycKey(<key>)` method to enable this `<key>` to grant (or revoke) the KYC status of other accounts so they can transact with your token. You would consider using the KYC flag when you need your token to be used only within parties that have been “authorized” to use it. For instance, known registered users or those who have passed identity verification. Think of this as identity and compliance features like anti-money laundering (AML) requirements or any type of off-ledger authentication mechanism, like if a user has signed up for your application.

The `.setKycKey(<key>)` method is not required when you create your token, so if you don’t use the method that means anyone who is associated with your token can transact without having to be “authorized”. No KYC key also means that KYC grant or revoke operations are not possible for the token in the future.

Enable Token KYC on an Account

We will continue with the NFT example from Part 1. However, we must create a new token using `.setKycKey(<key>)`. Before users can transfer the newly created token, we must grant KYC to those users, namely Alice and Bob.

```
{% code title="nft-part2.js" %}
javascript
// ENABLE TOKEN KYC FOR ALICE AND BOB
let [aliceKycRx, aliceKycTxId] = await kycEnableFcn(aliceId);
let [bobKyc, bobKycTxId] = await kycEnableFcn(bobId);

console.log(\n- Enabling token KYC for Alice's account: ${aliceKycRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${aliceKycTxId});
console.log(\n- Enabling token KYC for Bob's account: ${bobKyc.status});
console.log(- See: https://hashscan.io/${network}/transaction/${bobKycTxId});

{% endcode %}

javascript
// KYC ENABLE FUNCTION =====
async function kycEnableFcn(id) {
  let kycEnableTx = await new TokenGrantKycTransaction()
    .setAccountId(id)
    .setTokenId(tokenId)
    .freezeWith(client)
    .sign(kycKey);
  let kycSubmitTx = await kycEnableTx.execute(client);
  let kycRx = await kycSubmitTx.getReceipt(client);
  return [kycRx, kycSubmitTx.transactionId];
}
```

Console output:

```
bash
```

```
- Enabling token KYC for Alice's account: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.2520793@1723765780.358402849

- Enabling token KYC for Bob's account: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.2520793@1723765782.680313758
```

Disable Token KYC on an Account

After the KYC flag has been set to true for a user, the administrator, identity provider, or compliance manager can revoke or disable the KYC flag. After KYC is disabled for a user, he or she can no longer receive or send that token. Here's a sample code for disabling token KYC on Alice's account:

```
{% code title="nft-part2.js" %}
javascript
// DISABLE TOKEN KYC FOR ALICE
let kycDisableTx = await new TokenRevokeKycTransaction()
    .setAccountId(aliceId)
    .setTokenId(tokenId)
    .freezeWith(client)
    .sign(kycKey);
let kycDisableSubmitTx = await kycDisableTx.execute(client);
let kycDisableRx = await kycDisableSubmitTx.getReceipt(client);

console.log(\n- Disabling token KYC for Alice's account: $
{kycDisableRx.status});
console.log(
  - See: https://hashscan.io/${network}/transaction/$
{kycDisableSubmitTx.transactionId}
);

{% endcode %}
```

Console output:

```
bash
```

```
- Disabling token KYC for Alice's account: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.46446@1723796731.003707280
```

```
{% hint style="info" %}
```

Note: The following sections require Alice to have token KYC enabled.

```
{% endhint %}
```

Updating tokens

If you create a token using the `.setAdminKey(<key>)` method, then you can "update" that token, meaning change its metadata and characteristics. For instance, you can change the token name, symbol, or the keys that are associated with its controlled mutability. You could create a token that initially has a 1-to-1 key for minting and burning and, over time, change this to a threshold or multi-signature key. You can rotate the keys associated with compliance and administration or even remove them entirely, offering a more decentralized approach over time.

On the other hand, if you create a token without using `.setAdminKey(<key>)`, that token is immutable, and its properties cannot be modified.

In our example, we start by checking the initial KYC key for the token, then we

update the KYC key from \<kycKey> to \<newKycKey>, and then we query the token again to make sure the key change took place.

```
{% code title="nft-part2.js" %}
javascript
// QUERY TO CHECK INITIAL KYC KEY
var tokenInfo = await tQueryFcn();
console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

// UPDATE TOKEN PROPERTIES: NEW KYC KEY
let tokenUpdateTx = await new TokenUpdateTransaction()
    .setTokenId(tokenId)
    .setKycKey(newKycKey.publicKey)
    .freezeWith(client)
    .sign(adminKey);
let tokenUpdateSubmitTx = await tokenUpdateTx.execute(client);
let tokenUpdateRx = await tokenUpdateSubmitTx.getReceipt(client);
console.log(\n- Token update transaction (new KYC key): $
{tokenUpdateRx.status});
console.log(
  - See: https://hashscan.io/${network}/transaction/$
  {tokenUpdateSubmitTx.transactionId}
);

// QUERY TO CHECK CHANGE IN KYC KEY
var tokenInfo = await tQueryFcn();
console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

{% endcode %}

javascript
// TOKEN QUERY FUNCTION =====
async function tQueryFcn() {
  var tokenInfo = await new
TokenInfoQuery().setTokenId(tokenId).execute(client);
  return tokenInfo;
}
```

Console output:

```
bash
- KYC key for the NFT is:
302a300506032b65700321009178c6063f1ba8a61bf7e9674edf2aee06d547e0c18d0bb

- Token update transaction (new KYC key): SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.2520793@1723765784.825417112

- KYC key for the NFT is:
302a300506032b65700321001bfd2a1a63e5b7e5e159ca0911db29fb6013074b1652a253
```

Schedule Transactions

Scheduled transactions enable you to collect the required signatures for a transaction in preparation for its execution. This can be useful if you don't have all the required signatures for the network to immediately process the transaction. Currently, you can schedule: `TransferTransaction()` (for hbar and HTS tokens), `TokenMintTransaction()`, `TokenBurnTransaction()`, and `TopicMessageSubmitTransaction()`. More transactions are supported with new releases.

Now, we will schedule a token transfer from Bob to Alice using scheduled transactions. This token transfer requires signatures from both parties.

Given that Alice's and Bob's signatures are not immediately available (for the purposes of this example), we first create the NFT transfer without signatures. Then, we create the scheduled transaction using the constructor `ScheduleCreateTransaction()` and specify the NFT transfer as the transaction to schedule using the `.setScheduledTransaction()` method.

```
{% code title="nft-part2.js" %}
javascript
// CREATE THE NFT TRANSFER FROM BOB->ALICE TO BE SCHEDULED
// REQUIRES ALICE'S AND BOB'S SIGNATURES
let txToSchedule = new TransferTransaction()
  .addNftTransfer(tokenId, 2, bobId, aliceId)
  .addHbarTransfer(aliceId, nftPrice.negated())
  .addHbarTransfer(bobId, nftPrice);

// SCHEDULE THE NFT TRANSFER TRANSACTION CREATED IN THE LAST STEP
let scheduleTx = await new ScheduleCreateTransaction()
  .setScheduledTransaction(txToSchedule)
  .execute(client);
let scheduleRx = await scheduleTx.getReceipt(client);
let scheduleId = scheduleRx.scheduleId;
let scheduledTxId = scheduleRx.scheduledTransactionId;
console.log(- The schedule ID is: ${scheduleId});
console.log(- The scheduled transaction ID is: ${scheduledTxId} \n);

{% endcode %}
```

Console output:

```
bash
- The schedule ID is: 0.0.3071457
- The scheduled transaction ID is: 0.0.2520793@1723765788.746525125?scheduled
```

The token transfer is now scheduled, and it will be executed as soon as all required signatures are submitted. Note that the scheduled transaction IDs (scheduledTxId in this case) have a "scheduled" flag that you can use to confirm the status of the transaction.

As of the time of this writing, a scheduled transaction has 30 minutes to collect all the required signatures before it can be executed or it expires (deleted from the network). If you set an `<AdminKey>` for the scheduled transaction, then you can delete it before its execution or expiration.

Now, we submit the required signatures and get schedule information to check the status of our transfer.

```
{% code title="nft-part2.js" %}
javascript
// SUBMIT ALICE'S SIGNATURE FOR THE TRANSFER TRANSACTION
let aliceSignTx = await new ScheduleSignTransaction()
  .setScheduleId(scheduleId)
  .freezeWith(client)
  .sign(aliceKey);
let aliceSignSubmit = await aliceSignTx.execute(client);
let aliceSignRx = await aliceSignSubmit.getReceipt(client);
console.log(- Status of Alice's signature submission: ${aliceSignRx.status});
console.log(
  - See: https://hashscan.io/\${network}/transaction/\${aliceSignSubmit.transactionId}
);
```

```
// QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN ADDED)
```

```

scheduleQuery = await new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);
console.log(
    - Schedule triggered (all required signatures received): $
    {scheduleQuery.executed !== null}
);

// SUBMIT BOB'S SIGNATURE FOR THE TRANSFER TRANSACTION
let bobSignTx = await new ScheduleSignTransaction()
    .setScheduleId(scheduleId)
    .freezeWith(client)
    .sign(bobKey);
let bobSignSubmit = await bobSignTx.execute(client);
let bobSignRx = await bobSignSubmit.getReceipt(client);
console.log(- Status of Bob's signature submission: ${bobSignRx.status});
console.log(
    - See: https://hashscan.io/${network}/transaction/${
    bobSignSubmit.transactionId}
);

// QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN ADDED)
scheduleQuery = await new ScheduleInfoQuery()
    .setScheduleId(scheduleId)
    .execute(client);
console.log(
    \n- Schedule triggered (all required signatures received): ${
    scheduleQuery.executed !== null
    }
);

{% encode %}

```

Console output:

```

bash
- Status of Alice's signature submission: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.2520793@@1723765792.396881487

- Schedule triggered (all required signatures received): false

- Status of Bob's signature submission: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.2520793@@1723765789.984805782

- Schedule triggered (all required signatures received): true

```

The scheduled transaction was executed. It is still a good idea to verify that the transfer happened as we expected, so we check all the balances once more to confirm.

```

{% code title="nft-part2.js" %}
javascript
// VERIFY THAT THE SCHEDULED TRANSACTION (TOKEN TRANSFER) EXECUTED
oB = await bCheckerFcn(treasuryId);
aB = await bCheckerFcn(aliceId);
bB = await bCheckerFcn(bobId);
console.log(- Treasury balance: ${oB[0]} NFTs of ID: ${tokenId} and ${oB[1]});
console.log(- Alice balance: ${aB[0]} NFTs of ID: ${tokenId} and ${aB[1]});
console.log(- Bob balance: ${bB[0]} NFTs of ID: ${tokenId} and ${bB[1]});

{% encode %}

javascript

```

```
// BALANCE CHECKER FUNCTION =====
async function bCheckerFcn(id) {
  balanceCheckTx = await new
AccountBalanceQuery().setAccountId(id).execute(client);
  return [balanceCheckTx.tokens.map.get(tokenId.toString()),
balanceCheckTx.hbars];
}
```

Console output:

```
<figure><figcaption></figcaption></figure>
```

Conclusion

In this article, you saw examples of the flexibility you get when you create and configure your tokens with HTS - in a transparent and cryptographically provable way. Take advantage of this flexibility to tackle entirely new opportunities in the tokenization industry!

<details>

<summary>Code Check ☒

```
{% code title="nft-part2.js" %}
javascript
console.clear();
require("dotenv").config();
```

```
const {
  AccountId,
  PrivateKey,
  Client,
  TokenCreateTransaction,
  TokenInfoQuery,
  TokenType,
  CustomRoyaltyFee,
  CustomFixedFee,
  Hbar,
  HbarUnit,
  TokenSupplyType,
  TokenMintTransaction,
  TokenBurnTransaction,
  TransferTransaction,
  AccountBalanceQuery,
  TokenAssociateTransaction,
  TokenUpdateTransaction,
  TokenGrantKycTransaction,
  TokenRevokeKycTransaction,
  ScheduleCreateTransaction,
  ScheduleSignTransaction,
  ScheduleInfoQuery,
  AccountCreateTransaction,
} = require("@hashgraph/sdk");
```

```
// CONFIGURE ACCOUNTS AND CLIENT, AND GENERATE accounts and client, and
generate needed keys
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromStringECDSA(process.env.OPERATORKEYHEX);
const network = process.env.NETWORK;
```

```
const client = Client.forNetwork(network).setOperator(operatorId, operatorKey);
client.setDefaultMaxTransactionFee(new Hbar(50));
```

```

client.setDefaultMaxQueryPayment(new Hbar(1));

async function main() {
  // CREATE NEW HEDERA ACCOUNTS TO REPRESENT OTHER USERS
  const initBalance = new Hbar(1);

  const treasuryKey = PrivateKey.generateECDSA();
  const [treasurySt, treasuryId] = await accountCreateFcn(treasuryKey,
initBalance, client);
  console.log(- Treasury's account: https://hashscan.io/testnet/account/$
{treasuryId});
  const aliceKey = PrivateKey.generateECDSA();
  const [aliceSt, aliceId] = await accountCreateFcn(aliceKey, initBalance,
client);
  console.log(- Alice's account: https://hashscan.io/testnet/account/$
{aliceId});
  const bobKey = PrivateKey.generateECDSA();
  const [bobSt, bobId] = await accountCreateFcn(bobKey, initBalance,
client);
  console.log(- Bob's account: https://hashscan.io/testnet/account/$
{bobId});

  // GENERATE KEYS TO MANAGE FUNCTIONAL ASPECTS OF THE TOKEN
  const supplyKey = PrivateKey.generateECDSA();
  const adminKey = PrivateKey.generateECDSA();
  const pauseKey = PrivateKey.generateECDSA();
  const freezeKey = PrivateKey.generateECDSA();
  const wipeKey = PrivateKey.generateECDSA();
  const kycKey = PrivateKey.generate();
  const newKycKey = PrivateKey.generate();

  // DEFINE CUSTOM FEE SCHEDULE
  let nftCustomFee = new CustomRoyaltyFee()
    .setNumerator(1)
    .setDenominator(10)
    .setFeeCollectorAccountId(treasuryId)
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1,
HbarUnit.Tinybar))); // 1 HBAR = 100,000,000 Tinybar

  // IPFS CONTENT IDENTIFIERS FOR WHICH WE WILL CREATE NFTs - SEE
uploadJsonToIpfs.js
  let CIDs = [

Buffer.from("ipfs://bafkreibr7cyxmy4iyckmlyzige4ywccyygomwrcn4ldcldacw3nxe3ikgq"
),

Buffer.from("ipfs://bafkreig73xgqp7wy7qvjwz33rp3nknxaxqlsb7v3id24poe2dath7pj5dhe"
),

Buffer.from("ipfs://bafkreigl7q4oaofxll3o2cc3e3q3ofqzu6puennmambpulxexo5sryc6e"
),

Buffer.from("ipfs://bafkreiaoswszev3uoukkepctzpnzw56ey6w3xscokvsvmfrqdzmyhas6fu"
),

Buffer.from("ipfs://bafkreih6cajqynaqwbrmiabk2jxpy56rpf25zvg5lbien73p5ysnpehyjm"
),
  ];

```



```

// CREATE NFT WITH CUSTOM FEE
let nftCreateTx = await new TokenCreateTransaction()
    .setTokenName("Fall Collection")
    .setTokenSymbol("LEAF")
    .setTokenType(TokenType.NonFungibleUnique)
    .setDecimals(0)
    .setInitialSupply(0)
    .setTreasuryAccountId(treasuryId)
    .setSupplyType(TokenSupplyType.Finite)
    .setMaxSupply(CIDs.length)
    .setCustomFees([nftCustomFee])
    .setAdminKey(adminKey.publicKey)
    .setSupplyKey(supplyKey.publicKey)
    .setKycKey(kycKey.publicKey)
    .setPauseKey(pauseKey.publicKey)
    .setFreezeKey(freezeKey.publicKey)
    .setWipeKey(wipeKey.publicKey)
    .freezeWith(client)
    .sign(treasuryKey);

let nftCreateTxSign = await nftCreateTx.sign(adminKey);
let nftCreateSubmit = await nftCreateTxSign.execute(client);
let nftCreateRx = await nftCreateSubmit.getReceipt(client);
let tokenId = nftCreateRx.tokenId;
console.log(\n- Created NFT with Token ID: ${tokenId});
console.log(- See: https://hashscan.io/${network}/transaction/${nftCreateSubmit.transactionId});

// TOKEN QUERY TO CHECK THAT THE CUSTOM FEE SCHEDULE IS ASSOCIATED WITH
NFT
var tokenInfo = await tQueryFcn();
console.log( );
console.table(tokenInfo.customFees[0]);

// MINT NEW BATCH OF NFTs - CAN MINT UP TO 10 NFT SERIALS IN A SINGLE
TRANSACTION
let [nftMintRx, mintTxId] = await tokenMinterFcn(CIDs);
console.log(\n- Mint ${CIDs.length} serials for NFT collection ${tokenId}:
${nftMintRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${mintTxId});

// BURN THE LAST NFT IN THE COLLECTION
let tokenBurnTx = await new
TokenBurnTransaction().setTokenId(tokenId).setSerials([CIDs.length]).freezeWith(
client).sign(supplyKey);
let tokenBurnSubmit = await tokenBurnTx.execute(client);
let tokenBurnRx = await tokenBurnSubmit.getReceipt(client);
console.log(\n- Burn NFT with serial ${CIDs.length}: $
{tokenBurnRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${tokenBurnSubmit.transactionId});

var tokenInfo = await tQueryFcn();
console.log(- Current NFT supply: ${tokenInfo.totalSupply});

// MANUAL ASSOCIATION FOR ALICE'S ACCOUNT
let associateAliceTx = await new
TokenAssociateTransaction().setAccountId(aliceId).setTokenIds([tokenId]).freezeWith(
client).sign(aliceKey);
let associateAliceTxSubmit = await associateAliceTx.execute(client);
let associateAliceRx = await associateAliceTxSubmit.getReceipt(client);
console.log(\n- Alice NFT manual association: ${associateAliceRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/

```

```

{associateAliceTxSubmit.transactionId});

    // MANUAL ASSOCIATION FOR BOB'S ACCOUNT
    let associateBobTx = await new
TokenAssociateTransaction().setAccountId(bobId).setTokenIds([tokenId]).freezeWith
h(client).sign(bobKey);
    let associateBobTxSubmit = await associateBobTx.execute(client);
    let associateBobRx = await associateBobTxSubmit.getReceipt(client);
    console.log(\n- Bob NFT manual association: ${associateBobRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{associateBobTxSubmit.transactionId});

    // PART 2.1 STARTS
=====
    console.log(\nPART 2.1 STARTS
=====);
    // ENABLE TOKEN KYC FOR ALICE AND BOB
    let [aliceKycRx, aliceKycTxId] = await kycEnableFcn(aliceId);
    let [bobKyc, bobKycTxId] = await kycEnableFcn(bobId);
    console.log(\n- Enabling token KYC for Alice's account: $
{aliceKycRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{aliceKycTxId});
    console.log(\n- Enabling token KYC for Bob's account: ${bobKyc.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{bobKycTxId});
    67898;

    // DISABLE TOKEN KYC FOR ALICE
    let kycDisableTx = await new
TokenRevokeKycTransaction().setAccountId(aliceId).setTokenId(tokenId).freezeWith
(client).sign(kycKey);
    // let kycDisableSubmitTx = await kycDisableTx.execute(client);
    // let kycDisableRx = await kycDisableSubmitTx.getReceipt(client);
    // console.log(\n- Disabling token KYC for Alice's account: $
{kycDisableRx.status});
    // console.log(- See: https://hashscan.io/${network}/transaction/$
{kycDisableSubmitTx.transactionId});

    // QUERY TO CHECK INTIAL KYC KEY
    var tokenInfo = await tQueryFcn();
    console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

    // UPDATE TOKEN PROPERTIES: NEW KYC KEY
    let tokenUpdateTx = await new
TokenUpdateTransaction().setTokenId(tokenId).setKycKey(newKycKey.publicKey).free
zeWith(client).sign(adminKey);
    let tokenUpdateSubmitTx = await tokenUpdateTx.execute(client);
    let tokenUpdateRx = await tokenUpdateSubmitTx.getReceipt(client);
    console.log(\n- Token update transaction (new KYC key): $
{tokenUpdateRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenUpdateSubmitTx.transactionId});

    // QUERY TO CHECK CHANGE IN KYC KEY
    var tokenInfo = await tQueryFcn();
    console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

    // PART 2.1 ENDS
=====
    console.log(\nPART 2.1 ENDS
=====);

    // BALANCE CHECK 1

```

```

    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID: ${tokenId} and $
{oB[1]});
    console.log(- Alice balance: ${aB[0]} NFTs of ID: ${tokenId} and $
{aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID: ${tokenId} and ${bB[1]});

    // 1st TRANSFER NFT Treasury -> Alice
    let tokenTransferTx = await new
TransferTransaction().addNftTransfer(tokenId, 2, treasuryId,
aliceId).freezeWith(client).sign(treasuryKey);
    let tokenTransferSubmit = await tokenTransferTx.execute(client);
    let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);
    console.log(\n- NFT transfer Treasury -> Alice status: $
{tokenTransferRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenTransferSubmit.transactionId});

    // BALANCE CHECK 2
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and $
{oB[1]});
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    // 2nd NFT TRANSFER NFT Alice - >Bob
    let nftPrice = new Hbar(100000000, HbarUnit.Tinybar); // 1 HBAR =
10,000,000 Tinybar

    let tokenTransferTx2 = await new TransferTransaction()
        .addNftTransfer(tokenId, 2, aliceId, bobId)
        .addHbarTransfer(aliceId, nftPrice)
        .addHbarTransfer(bobId, nftPrice.negated())
        .freezeWith(client)
        .sign(aliceKey);
    let tokenTransferTx2Sign = await tokenTransferTx2.sign(bobKey);
    let tokenTransferSubmit2 = await tokenTransferTx2Sign.execute(client);
    let tokenTransferRx2 = await tokenTransferSubmit2.getReceipt(client);
    console.log(\n- NFT transfer Alice -> Bob status: $
{tokenTransferRx2.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenTransferSubmit2.transactionId});

    // BALANCE CHECK 3
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and $
{oB[1]});
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    // PART 2.2 STARTS
    =====
    console.log(\nPART 2.2 STARTS
    =====);

    // CREATE THE NFT TRANSFER FROM BOB -> ALICE TO BE SCHEDULED
    // REQUIRES ALICE'S AND BOB'S SIGNATURES
    let txToSchedule = new TransferTransaction()

```

```

        .addNftTransfer(tokenId, 2, bobId, aliceId)
        .addHbarTransfer(aliceId, nftPrice.negated())
        .addHbarTransfer(bobId, nftPrice);

    // SCHEDULE THE NFT TRANSFER TRANSACTION CREATED IN THE LAST STEP
    let scheduleTx = await new
ScheduleCreateTransaction().setScheduledTransaction(txToSchedule).execute(client
);
    let scheduleRx = await scheduleTx.getReceipt(client);
    let scheduleId = scheduleRx.scheduleId;
    let scheduledTxId = scheduleRx.scheduledTransactionId;
    console.log(\n- The schedule ID is: ${scheduleId});
    console.log(- The scheduled transaction ID is: ${scheduledTxId});

    // SUBMIT ALICE'S SIGNATURE FOR THE TRANSFER TRANSACTION
    let aliceSignTx = await new
ScheduleSignTransaction().setScheduleId(scheduleId).freezeWith(client).sign(alic
eKey);
    let aliceSignSubmit = await aliceSignTx.execute(client);
    let aliceSignRx = await aliceSignSubmit.getReceipt(client);
    console.log(\n- Status of Alice's signature submission: $
{aliceSignRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{aliceSignSubmit.transactionId});

    // QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN
ADDED)
    scheduleQuery = await new
ScheduleInfoQuery().setScheduleId(scheduleId).execute(client);
    console.log(\n- Schedule triggered (all required signatures received): $
{scheduleQuery.executed !== null});

    // SUBMIT BOB'S SIGNATURE FOR THE TRANSFER TRANSACTION
    let bobSignTx = await new
ScheduleSignTransaction().setScheduleId(scheduleId).freezeWith(client).sign(bobK
ey);
    let bobSignSubmit = await bobSignTx.execute(client);
    let bobSignRx = await bobSignSubmit.getReceipt(client);
    console.log(\n- Status of Bob's signature submission: $
{bobSignRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{bobSignSubmit.transactionId});

    // QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN
ADDED)
    scheduleQuery = await new
ScheduleInfoQuery().setScheduleId(scheduleId).execute(client);
    console.log(\n- Schedule triggered (all required signatures received): $
{scheduleQuery.executed !== null});

    // VERIFY THAT THE SCHEDULED TRANSACTION (TOKEN TRANSFER) EXECUTED
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID: ${tokenId} and $
{oB[1]});
    console.log(- Alice balance: ${aB[0]} NFTs of ID: ${tokenId} and $
{aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID: ${tokenId} and ${bB[1]});

    console.log(\n- THE END
=====);
    console.log(\n- 🖱 Go to:);
    console.log(- 🌐 www.hedera.com/discord\n);

```

```

client.close();

// ACCOUNT CREATOR FUNCTION =====
async function accountCreateFcn(pvKey, iBal, client) {
    const response = await new AccountCreateTransaction()
        .setInitialBalance(iBal)
        .setKey(pvKey.publicKey)
        .setMaxAutomaticTokenAssociations(10)
        .execute(client);
    const receipt = await response.getReceipt(client);
    return [receipt.status, receipt.accountId];
}

// TOKEN MINTER FUNCTION =====
async function tokenMinterFcn(CIDs) {
    let mintTx = new
TokenMintTransaction().setTokenId(tokenId).setMetadata(CIDs).freezeWith(client);
    let mintTxSign = await mintTx.sign(supplyKey);
    let mintTxSubmit = await mintTxSign.execute(client);
    let mintRx = await mintTxSubmit.getReceipt(client);
    return [mintRx, mintTxSubmit.transactionId];
}

// BALANCE CHECKER FUNCTION =====
async function bCheckerFcn(id) {
    balanceCheckTx = await new
AccountBalanceQuery().setAccountId(id).execute(client);
    return [balanceCheckTx.tokens.map.get(tokenId.toString()),
balanceCheckTx.hbars];
}

// KYC ENABLE FUNCTION =====
async function kycEnableFcn(id) {
    let kycEnableTx = await new
TokenGrantKycTransaction().setAccountId(id).setTokenId(tokenId).freezeWith(client).sign(kycKey);
    let kycSubmitTx = await kycEnableTx.execute(client);
    let kycRx = await kycSubmitTx.getReceipt(client);
    return [kycRx, kycSubmitTx.transactionId];
}


// TOKEN QUERY FUNCTION =====
async function tQueryFcn() {
    var tokenInfo = await new
TokenInfoQuery().setTokenId(tokenId).execute(client);
    return tokenInfo;
}
}
main();


```

{% endcode %}

</details>

Additional Resources

 [Project Repository](#)

 [Have a question? Ask on StackOverflow](#)

<table data-card-size="large" data-view="cards"><thead><tr><th align="center"></th><th data-hidden data-card-target

<p>Writer: Ed, DevRel Engineer</p> <p> GitHub LinkedIn </p>	<p>Editor: Krystal, Technical Writer</p> <p> GitHub Twitter </p>

hedera-token-service-part-3-how-to-pause-freeze-wipe-and-delete-nfts.md:

Hedera Token Service - Part 3: How to Pause, Freeze, Wipe, and Delete NFTs

In Part 1 of the series, you saw how to mint and transfer an NFT using the Hedera Token Service (HTS). In Part 2, you saw how to enable and disable token Know Your Customer (KYC), update token properties (if a token is mutable), and schedule transactions. In Part 3, you will learn how to use HTS capabilities that help you manage your tokens. Specifically, you will learn how to:

- Pause a token (stops all operations for a token ID)
- Freeze an account (stops all token operations only for a specific account)
- Wipe a token (wipe a partial or entire token balance for a specific account)
- Delete a token (the token will remain on the ledger)

{% embed url="https://youtu.be/8FWcsbm0udI" %}

Prerequisites

We recommend you complete the following introduction to get a basic understanding of Hedera transactions. This example does not build upon the previous examples.

- Get a Hedera testnet account.
- Set up your environment here.

☑ If you want the entire code used for this tutorial, skip to the Code Check section below.

Pause a Token

The pause transaction prevents a token from being involved in any kind of operation across all accounts. Specifying a `\<pauseKey>` during the creation of a token is a requirement to be able to pause token operations. The code below shows you that this key must sign the pause transaction. Note that you can't pause a token if it doesn't have a pause key. Also keep in mind that if this key was not set during token creation, then a token update to add this key is not possible.

Pausing a token may be useful in cases where a third party requests that you, as the administrator of a token, stop all operations for that token while something like an audit is conducted. The pause transaction provides you with a way to comply with requests of that nature.

In our example below, we pause the token, test that by trying a token transfer and checking the token `pauseStatus`, and then we unpauses the token to enable operations again.

```
{% code title="nft-part3.js" %}
javascript
// PAUSE ALL TOKEN OPERATIONS
let tokenPauseTx = await new TokenPauseTransaction()
```

```

        .setTokenId(tokenId)
        .freezeWith(client)
        .sign(pauseKey);
let tokenPauseSubmitTx = await tokenPauseTx.execute(client);
let tokenPauseRx = await tokenPauseSubmitTx.getReceipt(client);
console.log(- Token pause: ${tokenPauseRx.status});
console.log(
  - See: https://hashscan.io/${network}/transaction/${tokenPauseSubmitTx.transactionId}
);

// TEST THE TOKEN PAUSE BY TRYING AN NFT TRANSFER (TREASURY -> ALICE)
let tokenTransferTx3 = await new TransferTransaction()
  .addNftTransfer(tokenId, 3, treasuryId, aliceId)
  .freezeWith(client)
  .sign(treasuryKey);
let tokenTransferSubmit3 = await tokenTransferTx3.execute(client);
try {
  let tokenTransferRx3 = await tokenTransferSubmit3.getReceipt(client);
  console.log(
    \n-NFT transfer Treasury -> Alice status: ${tokenTransferRx3.status}
  );
} catch {
  // TOKEN QUERY TO CHECK PAUSE
  var tokenInfo = await tQueryFcn();
  console.log(
    \n- NFT transfer unsuccessful: Token ${tokenId} is paused (
    ${tokenInfo.pauseStatus})
  );
  console.log(
    - See: https://hashscan.io/${network}/transaction/${tokenTransferSubmit3.transactionId}
  );
}

// UNPAUSE ALL TOKEN OPERATIONS
let tokenUnpauseTx = await new TokenUnpauseTransaction()
  .setTokenId(tokenId)
  .freezeWith(client)
  .sign(pauseKey);
let tokenUnpauseSubmitTx = await tokenUnpauseTx.execute(client);
let tokenUnpauseRx = await tokenUnpauseSubmitTx.getReceipt(client);
console.log(- Token unpause: ${tokenUnpauseRx.status}\n);
console.log(
  - See: https://hashscan.io/${network}/transaction/${tokenUnpauseSubmitTx.transactionId}
);

{% endcode %}

javascript
// TOKEN QUERY FUNCTION =====
async function tQueryFcn() {
  var tokenInfo = await new
TokenInfoQuery().setTokenId(tokenId).execute(client);
  return tokenInfo;
}

```

Console output:

```

bash
- Token pause: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.46446@1723772395.193039130

```

- NFT transfer unsuccessful: Token 0.0.4686491 is paused (true)
- See: <https://hashscan.io/testnet/transaction/0.0.46446@1723772400.030831515>
- Token unpause: SUCCESS
- See: <https://hashscan.io/testnet/transaction/0.0.46446@1723772398.843664980>

Freeze a Token

Freezing a token stops "freezes" transfers of that token for a specific account ID. Note that this transaction must be signed by the `<freezeKey>` of the token. Once a freeze executes, the specified account is marked as "Frozen" and will not be able to receive or send tokens unless unfrozen.

In our example below, we first freeze Alice's account for the token ID we're working with, test the freeze by trying a token transfer, and then unfreeze Alice's account so she can transact the token again.

```
{% code title="nft-part3.js" %}
javascript
// FREEZE ALICE'S ACCOUNT FOR THIS TOKEN
let tokenFreezeTx = await new TokenFreezeTransaction()
    .setTokenId(tokenId)
    .setAccountId(aliceId)
    .freezeWith(client)
    .sign(freezeKey);
let tokenFreezeSubmitTx = await tokenFreezeTx.execute(client);
let tokenFreezeRx = await tokenFreezeSubmitTx.getReceipt(client);
console.log(
    \n- Freeze Alice's account for token ${tokenId}: ${tokenFreezeRx.status}
);
console.log(
    - See: https://hashscan.io/${network}/transaction/${
tokenFreezeSubmitTx.transactionId
}
);

// TEST THE TOKEN FREEZE FOR THE ACCOUNT BY TRYING A TRANSFER (ALICE -> BOB)
let tokenTransferTx4 = await new TransferTransaction()
    .addNftTransfer(tokenId, 2, aliceId, bobId)
    .addHbarTransfer(aliceId, nftPrice)
    .addHbarTransfer(bobId, nftPrice.negated())
    .freezeWith(client)
    .sign(aliceKey);
let tokenTransferTx4Sign = await tokenTransferTx4.sign(bobKey);
let tokenTransferSubmit4 = await tokenTransferTx4Sign.execute(client);
try {
    let tokenTransferRx4 = await tokenTransferSubmit4.getReceipt(client);
    console.log(
        \n- NFT transfer Alice -> Bob status: ${tokenTransferRx4.status}
    );
} catch {
    console.log(
        \n- NFT transfer Alice -> Bob unsuccessful: Alice's account is frozen for
this token
    );
    console.log(
        - See: https://hashscan.io/${network}/transaction/${
tokenTransferSubmit4.transactionId
}
    );
}

// UNFREEZE ALICE'S ACCOUNT FOR THIS TOKEN
```



```

let tokenUnfreezeTx = await new TokenUnfreezeTransaction()
    .setTokenId(tokenId)
    .setAccountId(aliceId)
    .freezeWith(client)
    .sign(freezeKey);
let tokenUnfreezeSubmitTx = await tokenUnfreezeTx.execute(client);
let tokenUnfreezeRx = await tokenUnfreezeSubmitTx.getReceipt(client);
console.log(
    \n- Unfreeze Alice's account for token ${tokenId}: ${tokenUnfreezeRx.status}
);
console.log(
    - See: https://hashscan.io/${network}/transaction/${
        tokenUnfreezeSubmitTx.transactionId
    }
);

{% endcode %}

```

Console output:

```

bash
- Freeze Alice's account for token 0.0.46864: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.46446@1723772402.841848773

- NFT transfer Alice -> Bob unsuccessful: Alice's account is frozen for this
token
- See: https://hashscan.io/testnet/transaction/0.0.46446@1723772405.380352596

- Unfreeze Alice's account for token 0.0.4686491: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.46446@1723772403.655969673

```

Wipe a Token

This operation wipes the provided amount of fungible or non-fungible tokens from the specified account. You see from the code below that this transaction must be signed by the token's `wipeKey`.

```

{% hint style="info" %}
Note: Wiping an account's tokens burns the tokens and decreases the total
supply. This transaction does not delete tokens from the treasury account. You
must use the Token Burn operation to delete tokens from the treasury.
{% endhint %}

```

In this case, we wipe the NFT that Alice currently holds. We then check Alice's balance and the NFT supply to see how these change with the wipe operation (these two values before the wipe are provided for comparison – see Part 2 for the details).

```

{% code title="nft-part3.js" %}
javascript
// WIPE THE TOKEN FROM ALICE'S ACCOUNT
let tokenWipeTx = await new TokenWipeTransaction()
    .setAccountId(aliceId)
    .setTokenId(tokenId)
    .setSerials([2])
    .freezeWith(client)
    .sign(wipeKey);
let tokenWipeSubmitTx = await tokenWipeTx.execute(client);
let tokenWipeRx = await tokenWipeSubmitTx.getReceipt(client);
console.log(\n- Wipe token ${tokenId} from Alice's account: $
{tokenWipeRx.status});
console.log(
    - See: https://hashscan.io/${network}/transaction/$

```

```

{tokenWipeSubmitTx.transactionId}
);

// CHECK ALICE'S BALANCE
aB = await bCheckerFcn(aliceId);
console.log(\n- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});

// TOKEN QUERY TO CHECK TOTAL TOKEN SUPPLY
var tokenInfo = await tQueryFcn();
console.log(- Current NFT supply: ${tokenInfo.totalSupply});

{% endcode %}

```

Console output:

```

<figure><figcaption></figcaption></figure>

```

Delete a Token

After you delete a token, it's no longer possible to perform any operations for that token, and transactions resolve to the error `TOKENWASDELETED`. Note that the token remains in the ledger, and you can still retrieve some information about it.

The delete operation must be signed by the token `\<adminKey>`. Remember from Part 1 that if this key is not set during token creation, then the token is immutable and deletion is not possible.

In our example, we delete the token and perform a query to double-check that the deletion was successful. Note that for NFTs, you can't delete a specific serial ID. Instead, you delete the entire class of the NFT specified by the token ID.

```

{% code title="nft-part3.js" %}
javascript
// DELETE THE TOKEN
let tokenDeleteTx = await new TokenDeleteTransaction()
  .setTokenId(tokenId)
  .freezeWith(client);
let tokenDeleteSign = await tokenDeleteTx.sign(adminKey);
let tokenDeleteSubmit = await tokenDeleteSign.execute(client);
let tokenDeleteRx = await tokenDeleteSubmit.getReceipt(client);
console.log(\n- Delete token ${tokenId}: ${tokenDeleteRx.status});
console.log(
  - See: https://hashscan.io/${network}/transaction/${
    tokenDeleteSubmitTx.transactionId
  }
);

// TOKEN QUERY TO CHECK DELETION
var tokenInfo = await tQueryFcn();
console.log(- Token ${tokenId} is deleted: ${tokenInfo.isDeleted});

{% endcode %}

```

Console output:

```

bash
- Delete token 0.0.4686491: SUCCESS
- See: https://hashscan.io/testnet/transaction/0.0.4644601@1723772409.964607591

- Token 0.0.4686491 is deleted: true


```

Conclusion

In this article, you saw key capabilities to help you manage your HTS tokens, including how to: pause, freeze, wipe, and delete tokens. If you haven't already, check out Part 1 and Part 2 of this tutorial series to see examples of how to do even MORE with HTS – you will see how to mint NFTs, transfer NFTs, perform token KYC, schedule transactions, and more.

Continue learning more in our learning center!

<details>

<summary>Code Check </summary>

```
{% code title="nft-part3.js" %}
javascript
console.clear();
require("dotenv").config();
```

```
const {
  AccountId,
  PrivateKey,
  Client,
  TokenCreateTransaction,
  TokenInfoQuery,
  TokenType,
  CustomRoyaltyFee,
  CustomFixedFee,
  Hbar,
  HbarUnit,
  TokenSupplyType,
  TokenMintTransaction,
  TokenBurnTransaction,
  TransferTransaction,
  AccountBalanceQuery,
  AccountUpdateTransaction,
  TokenAssociateTransaction,
  TokenUpdateTransaction,
  TokenGrantKycTransaction,
  TokenRevokeKycTransaction,
  ScheduleCreateTransaction,
  ScheduleSignTransaction,
  ScheduleInfoQuery,
  TokenPauseTransaction,
  TokenUnpauseTransaction,
  TokenWipeTransaction,
  TokenFreezeTransaction,
  TokenUnfreezeTransaction,
  TokenDeleteTransaction,
  AccountCreateTransaction,
} = require("@hashgraph/sdk");
```

```
// CONFIGURE ACCOUNTS AND CLIENT, AND GENERATE accounts and client, and
generate needed keys
const operatorId = AccountId.fromString(process.env.OPERATORID);
const operatorKey = PrivateKey.fromStringECDSA(process.env.OPERATORKEYHEX);
const network = process.env.NETWORK;
```

```
const client = Client.forNetwork(network).setOperator(operatorId, operatorKey);
client.setDefaultMaxTransactionFee(new Hbar(50));
client.setDefaultMaxQueryPayment(new Hbar(1));
```

```
async function main() {
  // CREATE NEW HEDERA ACCOUNTS TO REPRESENT OTHER USERS
```

```

const initBalance = new Hbar(1);

const treasuryKey = PrivateKey.generateECDSA();
const [treasurySt, treasuryId] = await accountCreateFcn(treasuryKey,
initBalance, client);
console.log(- Treasury's account: https://hashscan.io/testnet/account/${
treasuryId});
const aliceKey = PrivateKey.generateECDSA();
const [aliceSt, aliceId] = await accountCreateFcn(aliceKey, initBalance,
client);
console.log(- Alice's account: https://hashscan.io/testnet/account/${
aliceId});
const bobKey = PrivateKey.generateECDSA();
const [bobSt, bobId] = await accountCreateFcn(bobKey, initBalance,
client);
console.log(- Bob's account: https://hashscan.io/testnet/account/${
bobId});

// GENERATE KEYS TO MANAGE FUNCTIONAL ASPECTS OF THE TOKEN
const supplyKey = PrivateKey.generateECDSA();
const adminKey = PrivateKey.generateECDSA();
const pauseKey = PrivateKey.generateECDSA();
const freezeKey = PrivateKey.generateECDSA();
const wipeKey = PrivateKey.generateECDSA();
const kycKey = PrivateKey.generate();
const newKycKey = PrivateKey.generate();

// DEFINE CUSTOM FEE SCHEDULE
let nftCustomFee = new CustomRoyaltyFee()
    .setNumerator(1)
    .setDenominator(10)
    .setFeeCollectorAccountId(treasuryId)
    .setFallbackFee(new CustomFixedFee().setHbarAmount(new Hbar(1,
HbarUnit.Tinybar))); // 1 HBAR = 100,000,000 Tinybar

// IPFS CONTENT IDENTIFIERS FOR WHICH WE WILL CREATE NFTs - SEE
uploadJsonToIpfs.js
let CIDs = [

Buffer.from("ipfs://bafkreibr7cyxmy4iyckmlyzige4ywccyygomwrcn4ldcldacw3nxe3ikgq"
),

Buffer.from("ipfs://bafkreig73xgqp7wy7qvjwz33rp3nqxaxqlsb7v3id24poe2dath7pj5dhe"
),

Buffer.from("ipfs://bafkreigl7q4oaioifxll3o2cc3e3q3ofqzu6puennmambpulxexo5sryc6e"
),

Buffer.from("ipfs://bafkreiaoswszev3uoukkepctzpnzw56ey6w3xscokvsvmfrqdzmyhas6fu"
),

Buffer.from("ipfs://bafkreih6cajqynaqwbrmiabk2jxpy56rpf25zvg5lbien73p5ysnpehyjm"
),

];

// CREATE NFT WITH CUSTOM FEE
let nftCreateTx = await new TokenCreateTransaction()
    .setTokenName("Fall Collection")
    .setTokenSymbol("LEAF")

```

```

        .setTokenType(TokenType.NonFungibleUnique)
        .setDecimals(0)
        .setInitialSupply(0)
        .setTreasuryAccountId(treasuryId)
        .setSupplyType(TokenSupplyType.Finite)
        .setMaxSupply(CIDs.length)
        .setCustomFees([nftCustomFee])
        .setAdminKey(adminKey.publicKey)
        .setSupplyKey(supplyKey.publicKey)
        .setKycKey(kycKey.publicKey)
        .setPauseKey(pauseKey.publicKey)
        .setFreezeKey(freezeKey.publicKey)
        .setWipeKey(wipeKey.publicKey)
        .freezeWith(client)
        .sign(treasuryKey);

let nftCreateTxSign = await nftCreateTx.sign(adminKey);
let nftCreateSubmit = await nftCreateTxSign.execute(client);
let nftCreateRx = await nftCreateSubmit.getReceipt(client);
let tokenId = nftCreateRx.tokenId;
console.log(\n- Created NFT with Token ID: ${tokenId});
console.log(- See: https://hashscan.io/${network}/transaction/${nftCreateSubmit.transactionId});

// TOKEN QUERY TO CHECK THAT THE CUSTOM FEE SCHEDULE IS ASSOCIATED WITH
NFT
var tokenInfo = await tQueryFcn();
console.log( );
console.table(tokenInfo.customFees[0]);

// MINT NEW BATCH OF NFTs - CAN MINT UP TO 10 NFT SERIALS IN A SINGLE
TRANSACTION
let [nftMintRx, mintTxId] = await tokenMinterFcn(CIDs);
console.log(\n- Mint ${CIDs.length} serials for NFT collection ${tokenId}:
${nftMintRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${mintTxId});

// BURN THE LAST NFT IN THE COLLECTION
let tokenBurnTx = await new
TokenBurnTransaction().setTokenId(tokenId).setSerials([CIDs.length]).freezeWith(
client).sign(supplyKey);
let tokenBurnSubmit = await tokenBurnTx.execute(client);
let tokenBurnRx = await tokenBurnSubmit.getReceipt(client);
console.log(\n- Burn NFT with serial ${CIDs.length}: $
{tokenBurnRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${tokenBurnSubmit.transactionId});

var tokenInfo = await tQueryFcn();
console.log(- Current NFT supply: ${tokenInfo.totalSupply});

// MANUAL ASSOCIATION FOR ALICE'S ACCOUNT
let associateAliceTx = await new
TokenAssociateTransaction().setAccountId(aliceId).setTokenIds([tokenId]).freezeWith(
client).sign(aliceKey);
let associateAliceTxSubmit = await associateAliceTx.execute(client);
let associateAliceRx = await associateAliceTxSubmit.getReceipt(client);
console.log(\n- Alice NFT manual association: ${associateAliceRx.status});
console.log(- See: https://hashscan.io/${network}/transaction/${associateAliceTxSubmit.transactionId});

// MANUAL ASSOCIATION FOR BOB'S ACCOUNT
let associateBobTx = await new

```

```

TokenAssociateTransaction().setAccountId(bobId).setTokenIds([tokenId]).freezeWith
h(client).sign(bobKey);
    let associateBobTxSubmit = await associateBobTx.execute(client);
    let associateBobRx = await associateBobTxSubmit.getReceipt(client);
    console.log(\n- Bob NFT manual association: ${associateBobRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/${
associateBobTxSubmit.transactionId});

    // PART 2.1 STARTS
=====
    console.log(\nPART 2.1 STARTS
=====);
    // ENABLE TOKEN KYC FOR ALICE AND BOB
    let [aliceKycRx, aliceKycTxId] = await kycEnableFcn(aliceId);
    let [bobKyc, bobKycTxId] = await kycEnableFcn(bobId);
    console.log(\n- Enabling token KYC for Alice's account: $
{aliceKycRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{aliceKycTxId});
    console.log(\n- Enabling token KYC for Bob's account: ${bobKyc.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{bobKycTxId});
    67898;

    // DISABLE TOKEN KYC FOR ALICE
    let kycDisableTx = await new
TokenRevokeKycTransaction().setAccountId(aliceId).setTokenId(tokenId).freezeWith
(client).sign(kycKey);
    // let kycDisableSubmitTx = await kycDisableTx.execute(client);
    // let kycDisableRx = await kycDisableSubmitTx.getReceipt(client);
    // console.log(\n- Disabling token KYC for Alice's account: $
{kycDisableRx.status});
    // console.log(- See: https://hashscan.io/${network}/transaction/$
{kycDisableSubmitTx.transactionId});

    // QUERY TO CHECK INTIAL KYC KEY
    var tokenInfo = await tQueryFcn();
    console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

    // UPDATE TOKEN PROPERTIES: NEW KYC KEY
    let tokenUpdateTx = await new
TokenUpdateTransaction().setTokenId(tokenId).setKycKey(newKycKey.publicKey).free
zeWith(client).sign(adminKey);
    let tokenUpdateSubmitTx = await tokenUpdateTx.execute(client);
    let tokenUpdateRx = await tokenUpdateSubmitTx.getReceipt(client);
    console.log(\n- Token update transaction (new KYC key): $
{tokenUpdateRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenUpdateSubmitTx.transactionId});

    // QUERY TO CHECK CHANGE IN KYC KEY
    var tokenInfo = await tQueryFcn();
    console.log(\n- KYC key for the NFT is: \n${tokenInfo.kycKey.toString()});

    // PART 2.1 ENDS
=====
    console.log(\nPART 2.1 ENDS
=====);

    // BALANCE CHECK 1
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID: ${tokenId} and $

```

```

{oB[1]}});
    console.log(- Alice balance: ${aB[0]} NFTs of ID: ${tokenId} and $
{aB[1]}});
    console.log(- Bob balance: ${bB[0]} NFTs of ID: ${tokenId} and ${bB[1]}});

    // 1st TRANSFER NFT Treasury -> Alice
    let tokenTransferTx = await new
TransferTransaction().addNftTransfer(tokenId, 2, treasuryId,
aliceId).freezeWith(client).sign(treasuryKey);
    let tokenTransferSubmit = await tokenTransferTx.execute(client);
    let tokenTransferRx = await tokenTransferSubmit.getReceipt(client);
    console.log(\n- NFT transfer Treasury -> Alice status: $
{tokenTransferRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenTransferSubmit.transactionId});

    // BALANCE CHECK 2
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and $
{oB[1]}});
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    // 2nd NFT TRANSFER NFT Alice - >Bob
    let nftPrice = new Hbar(100000000, HbarUnit.Tinybar); // 1 HBAR =
10,000,000 Tinybar

    let tokenTransferTx2 = await new TransferTransaction()
        .addNftTransfer(tokenId, 2, aliceId, bobId)
        .addHbarTransfer(aliceId, nftPrice)
        .addHbarTransfer(bobId, nftPrice.negated())
        .freezeWith(client)
        .sign(aliceKey);
    let tokenTransferTx2Sign = await tokenTransferTx2.sign(bobKey);
    let tokenTransferSubmit2 = await tokenTransferTx2Sign.execute(client);
    let tokenTransferRx2 = await tokenTransferSubmit2.getReceipt(client);
    console.log(\n- NFT transfer Alice -> Bob status: $
{tokenTransferRx2.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenTransferSubmit2.transactionId});

    // BALANCE CHECK 3
    oB = await bCheckerFcn(treasuryId);
    aB = await bCheckerFcn(aliceId);
    bB = await bCheckerFcn(bobId);
    console.log(\n- Treasury balance: ${oB[0]} NFTs of ID:${tokenId} and $
{oB[1]}});
    console.log(- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and ${aB[1]});
    console.log(- Bob balance: ${bB[0]} NFTs of ID:${tokenId} and ${bB[1]});

    // PART 2.2 STARTS
=====
    console.log(\nPART 2.2 STARTS
=====);

    // CREATE THE NFT TRANSFER FROM BOB -> ALICE TO BE SCHEDULED
    // REQUIRES ALICE'S AND BOB'S SIGNATURES
    let txToSchedule = new TransferTransaction()
        .addNftTransfer(tokenId, 2, bobId, aliceId)
        .addHbarTransfer(aliceId, nftPrice.negated())
        .addHbarTransfer(bobId, nftPrice);

```

```

// SCHEDULE THE NFT TRANSFER TRANSACTION CREATED IN THE LAST STEP
let scheduleTx = await new
ScheduleCreateTransaction().setScheduledTransaction(txToSchedule).execute(client
);
    let scheduleRx = await scheduleTx.getReceipt(client);
    let scheduleId = scheduleRx.scheduleId;
    let scheduledTxId = scheduleRx.scheduledTransactionId;
    console.log(\n- The schedule ID is: ${scheduleId});
    console.log(- The scheduled transaction ID is: ${scheduledTxId});

// SUBMIT ALICE'S SIGNATURE FOR THE TRANSFER TRANSACTION
let aliceSignTx = await new
ScheduleSignTransaction().setScheduleId(scheduleId).freezeWith(client).sign(alic
eKey);
    let aliceSignSubmit = await aliceSignTx.execute(client);
    let aliceSignRx = await aliceSignSubmit.getReceipt(client);
    console.log(\n- Status of Alice's signature submission: $
{aliceSignRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{aliceSignSubmit.transactionId});

// QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN
ADDED)
    scheduleQuery = await new
ScheduleInfoQuery().setScheduleId(scheduleId).execute(client);
    console.log(\n- Schedule triggered (all required signatures received): $
{scheduleQuery.executed !== null});

// SUBMIT BOB'S SIGNATURE FOR THE TRANSFER TRANSACTION
let bobSignTx = await new
ScheduleSignTransaction().setScheduleId(scheduleId).freezeWith(client).sign(bobK
ey);
    let bobSignSubmit = await bobSignTx.execute(client);
    let bobSignRx = await bobSignSubmit.getReceipt(client);
    console.log(\n- Status of Bob's signature submission: $
{bobSignRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{bobSignSubmit.transactionId});

// QUERY TO CONFIRM IF THE SCHEDULE WAS TRIGGERED (SIGNATURES HAVE BEEN
ADDED)
    scheduleQuery = await new
ScheduleInfoQuery().setScheduleId(scheduleId).execute(client);
    console.log(\n- Schedule triggered (all required signatures received): $
{scheduleQuery.executed !== null});

// VERIFY THAT THE SCHEDULED TRANSACTION (TOKEN TRANSFER) EXECUTED
oB = await bCheckerFcn(treasuryId);
aB = await bCheckerFcn(aliceId);
bB = await bCheckerFcn(bobId);
console.log(\n- Treasury balance: ${oB[0]} NFTs of ID: ${tokenId} and $
{oB[1]});
console.log(- Alice balance: ${aB[0]} NFTs of ID: ${tokenId} and $
{aB[1]});
console.log(- Bob balance: ${bB[0]} NFTs of ID: ${tokenId} and ${bB[1]});

// PART 3 =====
console.log(\nPART 3 STARTS
=====);

// PAUSE ALL TOKEN OEPRATIONS
let tokenPauseTx = await new
TokenPauseTransaction().setTokenId(tokenId).freezeWith(client).sign(pauseKey);
let tokenPauseSubmitTx = await tokenPauseTx.execute(client);

```



```

    let tokenPauseRx = await tokenPauseSubmitTx.getReceipt(client);
    console.log(\n- Token pause: ${tokenPauseRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/${tokenPauseSubmitTx.transactionId});

    // TEST THE TOKEN PAUSE BY TRYING AN NFT TRANSFER (TREASURY -> ALICE)
    let tokenTransferTx3 = await new
TransferTransaction().addNftTransfer(tokenId, 3, treasuryId,
aliceId).freezeWith(client).sign(treasuryKey);
    let tokenTransferSubmit3 = await tokenTransferTx3.execute(client);
    try {
        let tokenTransferRx3 = await
tokenTransferSubmit3.getReceipt(client);
        console.log(\n- NFT transfer Treasury -> Alice status: $
{tokenTransferRx3.status});
    } catch {
        // TOKEN QUERY TO CHECK PAUSE
        var tokenInfo = await tQueryFcn();
        console.log(\n- NFT transfer unsuccessful: Token ${tokenId} is
paused (${tokenInfo.pauseStatus});
        console.log(- See: https://hashscan.io/${network}/transaction/${tokenTransferSubmit3.transactionId});
    }

    // UNPAUSE ALL TOKEN OEPRATIONS
    let tokenUnpauseTx = await new
TokenUnpauseTransaction().setTokenId(tokenId).freezeWith(client).sign(pauseKey);
    let tokenUnpauseSubmitTx = await tokenUnpauseTx.execute(client);
    let tokenUnpauseRx = await tokenUnpauseSubmitTx.getReceipt(client);
    console.log(\n- Token unpause: ${tokenUnpauseRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/${tokenUnpauseSubmitTx.transactionId});

    // FREEZE ALICE'S ACCOUNT FOR THIS TOKEN
    let tokenFreezeTx = await new
TokenFreezeTransaction().setTokenId(tokenId).setAccountId(aliceId).freezeWith(cl
ient).sign(freezeKey);
    let tokenFreezeSubmitTx = await tokenFreezeTx.execute(client);
    let tokenFreezeRx = await tokenFreezeSubmitTx.getReceipt(client);
    console.log(\n- Freeze Alice's account for token ${tokenId}: $
{tokenFreezeRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/${tokenFreezeSubmitTx.transactionId});

    // TEST THE TOKEN FREEZE FOR THE ACCOUNT BY TRYING A TRANSFER (ALICE ->
BOB)
    let tokenTransferTx4 = await new TransferTransaction()
        .addNftTransfer(tokenId, 2, aliceId, bobId)
        .addHbarTransfer(aliceId, nftPrice)
        .addHbarTransfer(bobId, nftPrice.negated())
        .freezeWith(client)
        .sign(aliceKey);
    let tokenTransferTx4Sign = await tokenTransferTx4.sign(bobKey);
    let tokenTransferSubmit4 = await tokenTransferTx4Sign.execute(client);
    try {
        let tokenTransferRx4 = await
tokenTransferSubmit4.getReceipt(client);
        console.log(\n- NFT transfer Alice -> Bob status: $
{tokenTransferRx4.status});
    } catch {
        console.log(\n- NFT transfer Alice -> Bob unsuccessful: Alice's
account is frozen for this token);
        console.log(- See: https://hashscan.io/${network}/transaction/${tokenTransferSubmit4.transactionId});
    }

```

```

    }
    // UNFREEZE ALICE'S ACCOUNT FOR THIS TOKEN
    let tokenUnfreezeTx = await new
TokenUnfreezeTransaction().setTokenId(tokenId).setAccountId(aliceId).freezeWith(
client).sign(freezeKey);
    let tokenUnfreezeSubmitTx = await tokenUnfreezeTx.execute(client);
    let tokenUnfreezeRx = await tokenUnfreezeSubmitTx.getReceipt(client);
    console.log(\n- Unfreeze Alice's account for token ${tokenId}: $
{tokenUnfreezeRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenUnfreezeSubmitTx.transactionId});

    // WIPE THE TOKEN FROM ALICE'S ACCOUNT
    let tokenWipeTx = await new
TokenWipeTransaction().setAccountId(aliceId).setTokenId(tokenId).setSerials([2])
.freezeWith(client).sign(wipeKey);
    let tokenWipeSubmitTx = await tokenWipeTx.execute(client);
    let tokenWipeRx = await tokenWipeSubmitTx.getReceipt(client);
    console.log(\n- Wipe token ${tokenId} from Alice's account: $
{tokenWipeRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenWipeSubmitTx.transactionId});

    // CHECK ALICE'S BALANCE
    aB = await bCheckerFcn(aliceId);
    console.log(\n- Alice balance: ${aB[0]} NFTs of ID:${tokenId} and $
{aB[1]});

    // TOKEN QUERY TO CHECK TOTAL TOKEN SUPPLY
    var tokenInfo = await tQueryFcn();
    console.log(- Current NFT supply: ${tokenInfo.totalSupply});

    // DELETE THE TOKEN
    let tokenDeleteTx = new
TokenDeleteTransaction().setTokenId(tokenId).freezeWith(client);
    let tokenDeleteSign = await tokenDeleteTx.sign(adminKey);
    let tokenDeleteSubmitTx = await tokenDeleteSign.execute(client);
    let tokenDeleteRx = await tokenDeleteSubmitTx.getReceipt(client);
    console.log(\n- Delete token ${tokenId}: ${tokenDeleteRx.status});
    console.log(- See: https://hashscan.io/${network}/transaction/$
{tokenDeleteSubmitTx.transactionId});

    // TOKEN QUERY TO CHECK DELETION
    var tokenInfo = await tQueryFcn();
    console.log(\n- Token ${tokenId} is deleted: ${tokenInfo.isDeleted});

    console.log(\n- THE END
=====);
    console.log(\n- 🖱 Go to:);
    console.log(- 🌐 www.hedera.com/discord\n);

    client.close();

    // ACCOUNT CREATOR FUNCTION =====
    async function accountCreateFcn(pvKey, iBal, client) {
        const response = await new AccountCreateTransaction()
            .setInitialBalance(iBal)
            .setKey(pvKey.publicKey)
            .setMaxAutomaticTokenAssociations(10)
            .execute(client);
        const receipt = await response.getReceipt(client);
        return [receipt.status, receipt.accountId];
    }

```

```

// TOKEN MINTER FUNCTION =====
async function tokenMinterFcn(CIDs) {
  let mintTx = new
TokenMintTransaction().setTokenId(tokenId).setMetadata(CIDs).freezeWith(client);
  let mintTxSign = await mintTx.sign(supplyKey);
  let mintTxSubmit = await mintTxSign.execute(client);
  let mintRx = await mintTxSubmit.getReceipt(client);
  return [mintRx, mintTxSubmit.transactionId];
}

// BALANCE CHECKER FUNCTION =====
async function bCheckerFcn(id) {
  balanceCheckTx = await new
AccountBalanceQuery().setAccountId(id).execute(client);
  return [balanceCheckTx.tokens.map.get(tokenId.toString()),
balanceCheckTx.hbars];
}

// KYC ENABLE FUNCTION =====
async function kycEnableFcn(id) {
  let kycEnableTx = await new
TokenGrantKycTransaction().setAccountId(id).setTokenId(tokenId).freezeWith(client).sign(kycKey);
  let kycSubmitTx = await kycEnableTx.execute(client);
  let kycRx = await kycSubmitTx.getReceipt(client);
  return [kycRx, kycSubmitTx.transactionId];
}


// TOKEN QUERY FUNCTION =====
async function tQueryFcn() {
  var tokenInfo = await new
TokenInfoQuery().setTokenId(tokenId).execute(client);
  return tokenInfo;
}
}
main();

```

{% endcode %}

</details>

Additional Resources

 Project Repository

 Have a question? Ask on StackOverflow

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Ed, DevRel Engineer</p><p><a
href="https://github.com/ed-marquez">GitHub</a> | <a
href="https://www.linkedin.com/in/ed-marquez/">LinkedIn</a></p></td><td><a
href="https://www.linkedin.com/in/ed-marquez/">https://www.linkedin.com/in/ed-
marquez/</a></td></tr><tr><td align="center"><p>Editor: Krystal, Technical
Writer</p><p><a href="https://github.com/theekrystallee">GitHub</a> | <a
href="https://x.com/theekrystallee">Twitter</a></p></td><td><a
href="https://x.com/theekrystallee">https://x.com/theekrystallee</a></td></
tr></tbody></table>

```

README.md:

Tokens

structure-your-token-metadata-using-json-schema-v2.md:

Structure Your Token Metadata Using JSON Schema V2

Summary

When you create a new fungible or non-fungible token, you have the ability to add metadata. It's common to add metadata for NFTs, but also fungible tokens. The biggest problem with metadata is that it's often unstructured or doesn't follow a set of guidelines.

Therefore, Hedera has developed the "Token Metadata JSON Schema V2" for developers and creators who want to structure their metadata in an organized way. The biggest benefit of using this community-accepted standard is that most of the tooling on the Hedera network can scrape and interpret your metadata, like NFT explorers listing rarity attributes based on your metadata.

Prerequisites

We recommend you complete one of the two tutorials below that teach you how to create a fungible or non-fungible token on the Hedera network.

```
{% content-ref url="create-and-transfer-your-first-nft.md" %}  
create-and-transfer-your-first-nft.md  
{% endcontent-ref %}
```

```
{% content-ref url="create-and-transfer-your-first-fungible-token.md" %}  
create-and-transfer-your-first-fungible-token.md  
{% endcontent-ref %}
```

Table of Contents

1. Connect Metadata
2. Metadata Schema
3. Verify Metadata
4. Video Tutorial

How do you connect metadata to a token?

It's essential to understand that the token metadata JSON schema V2 requires you to store metadata using a storage solution, centralized or decentralized, such as IPFS or Arweave.

When creating a non-fungible token using the Hedera Token Service, you set the metadata value to the metadata JSON file to define your NFT, wherever it's stored. This technique allows you to connect the metadata to the token created on the Hedera network. The "memo" or "symbol" fields are not allowed on the NFT.

An excerpt from the NFT minting tutorial shows this connection when minting a new NFT.

```
{% tabs %}  
{% tab title="Java" %}
```

```

<pre class="language-java"><code class="lang-java"><strong>// IPFS content
identifier (CID) that points to your metadata
</strong>String CID = ("QmTzWcVfk88JRqjTpVwHzBeULRTNzHY7mnBSG42CpwHmPa") ;

// Mint a new NFT
TokenMintTransaction mintTx = new TokenMintTransaction()
    .setTokenId(tokenId)
    .addMetadata(CID.getBytes())
    .freezeWith(client);
</code></pre>
{% endtab %}

{% tab title="JavaScript" %}
javascript
// IPFS content identifier (CID) that points to your metadata
let CID = "ipfs://QmTzWcVfk88JRqjTpVwHzBeULRTNzHY7mnBSG42CpwHmPa";

// Mint new NFT
let mintTx = await new TokenMintTransaction()
    .setTokenId(tokenId)
    .setMetadata([Buffer.from(CID)])
    .freezeWith(client);

{% endtab %}

{% tab title="Go" %}
<pre class="language-go"><code class="lang-go"><strong>// IPFS content
identifier (CID) that points to your metadata
</strong><strong>CID := "QmTzWcVfk88JRqjTpVwHzBeULRTNzHY7mnBSG42CpwHmPa"
</strong>
//Mint new NFT
mintTx, err := hedera.NewTokenMintTransaction().
    SetTokenID(tokenId).
    SetMetadata([]byte(CID)).
    FreezeWith(client)
</code></pre>
{% endtab %}
{% endtabs %}

```

What does the Token Metadata JSON schema V2 look like?

First of all, you can find the full reference implementation of this JSON schema [here](#):

Hedera Improvement Proposals GitHub Repository
 NFT.Storage (gateway link)
 IPFS: [ipfs://bafkreidcsqzr5su356thecwuyzrhsgekfdsqzuyuqxtsu4vh7oc34iv5oy](https://ipfs.io/ipfs/bafkreidcsqzr5su356thecwuyzrhsgekfdsqzuyuqxtsu4vh7oc34iv5oy)

Let's take a look at the different fields you can specify.

Required fields:

```

{% tabs %}
{% tab title="name, type, and image" %}
The schema defines three required fields:

```

```

name: Full name of the NFT
type: MIME type for the image
image: A URI pointing to an image (decentralized or centralized storage).&#x20;

```

The image field can both serve as a preview or full-resolution image for your NFT to ensure cross-platform compatibility. The image field will be displayed in

wallets and marketplaces by default.

Creators are recommended to point to a thumbnail in the image field and put the high-resolution image in the files array with the isdefaultfile boolean set to indicate that this file represents the default image for the NFT. (This is shown in the next section)

Here's a small example of an implementation.

```
json
{
  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png"
}

{% endtab %}
{% endtabs %}
```

Optional fields:

```
{% tabs %}
{% tab title="files" %}
The files field represents an array containing file objects. For collectible
NFTs, the files array allows you to store the high-resolution image of your NFT.
However, you can also use this field for multi-file NFTs. Each file object
requires a URI and type.&#x20;
```

It's recommended to use the is\default\file field to indicate which file is the main file for your NFT. Besides that, the files array allows you to upload or link file-specific metadata. This allows you to nest files indefinitely.

```
json
{
  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png",
  "files": [
    {
      "uri": "https://myserver.com/high-resolution-nft-001.png",
      "checksum":
"9defbb6402d4bf39f2ea580099c73194647b24a659b6f6b778e3dd71755b8862",
      "isdefaultfile": true,
      "type": "image/png"
    }
  ]
}

{% endtab %}
```

```
{% tab title="properties" %}
Additional fields are not allowed to be defined at the root level of the
metadata object.&#x20;
```

```
json
// ✗ Not allowed to add "website" to the root of the object
{
  "website": "https://mysite.com",
  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png"
}
```

If you want to add custom fields, you can add them to the properties object. For example, you are linking to a website or social media pages. You can structure the metadata within the properties field as needed. Some developers even prefer defining their own standard for the properties field, not for the entire metadata object.

```
<pre class="language-json"><code class="lang-json"><strong>//  Good example
</strong><strong>{
</strong>  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png",
  "properties": {
    "website": "https://mysite.com",
    "socials": {
      "linkedin": "https://www.linkedin.com/in/myprofile/"
    }
  }
}
</code></pre>
```

{% endtab %}

{% tab title="attributes" %}

The attributes field is specifically used to calculate the rarity of NFTs. It's an industry-accepted way to define traits and their values for a collectible NFT collection in order to calculate the rarity score of the NFT.

The attributes field consists of an array of attribute objects. This is the structure of such an object:

trait\type: (required) Name of the trait.
value: (required) Value for the trait, e.g., for a traittype = clothing, values can be pants, shirt, or t-shirt.
display\type: (optional) Allows you to specify how the trait should be displayed. For instance, you can set it to datetime so the person or bot knows the value should be interpreted as a date.
max\value: (optional) It's possible to set a maxvalue for a numerical value.

```
<pre class="language-json"><code class="lang-json">{
  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png",
  "attributes": [
    {
      "traittype": "clothing",
      "value": "pants"
    },
    {
      "traittype": "color",
      "displaytype": "color",
      "value": "rgb(255,0,0)"
    },
    {
      "traittype": "hasPipe",
      "displaytype": "boolean",
      "value": true
    },
    {
      "traittype": "coolness",
      "displaytype": "boost",
      "value": 10,
      "maxvalue": 100
    }
  ]
}
</code></pre>
```

```

    },
    {
      "traittype": "stamina",
      "displaytype": "percentage",
      "value": 83
    },
    {
      "traittype": "birth",
      "displaytype": "datetime",
      "value": 732844800
    }
  ]
}
</code></pre>

```

Extra resources: You can find all information about the attributes field in the detailed schema specification.

```
{% endtab %}
```

```
{% tab title="localization" %}
```

The standard also allows for localization. Each locale links to another metadata file containing localized metadata and files. This allows for a clean metadata structure. Don't define a new localization object for a localized metadata file to avoid infinite looping when parsing an NFT's metadata file.

Note that the `localization.uri` property contains `{locale}`. The `{locale}` part references a locale in the `locales` array. You should use two-letter language codes according to the ISO 639-1 standard to define languages.

```

json
{
  "name": "My first NFT",
  "type": "image/png",
  "image": "https://myserver.com/preview-image-nft-001.png",
  "localization": {
    "uri":
"ipfs://QmWS1VAdMD353A6SDk9wNyvKT14kyCiZrNDYAad4w1tKqT/{locale}.json",
    "default": "en",
    "locales": ["es", "fr"]
  }
}

```

In this example, the default locale is set to English (en) and the schema provides localization for Spanish (es) and French (fr). The schema assumes it can find the Spanish and French version under the following URIs because of the way it has been specified using the interpolation notation `{locale}.json`. This is what the resulting URIs should look like:

```

ipfs://QmWS1VAdMD353A6SDk9wNyvKT14kyCiZrNDYAad4w1tKqT/es.json
ipfs://QmWS1VAdMD353A6SDk9wNyvKT14kyCiZrNDYAad4w1tKqT/fr.json

```

A localized file would have the same structure, but doesn't specify any localization field. Here's an example for the French version.

```

<pre class="language-json"><code class="lang-json"><strong>// metadata for
ipfs://QmWS1VAdMD353A6SDk9wNyvKT14kyCiZrNDYAad4w1tKqT/fr.json
</strong><strong>{
</strong>  "name": "Mon NFT (French)",
    "type": "image/png",
    "image": "https://myserver.com/preview-image-nft-001.png"
}

```



```
</code></pre>
{% endtab %}
```

{% tab title="description, creator, creatorDID, checksum, and format" %}
Here's a list of optional fields you can define according to the Token Metadata JSON Schema V2 specification.

description: A text that describes your token or NFT collection.
creator: Identifies the artist name(s).
creatorDID: Points to a decentralized identifier to identify the creator.
checksum: Cryptographic SHA-256 hash of the representation of the image resource or resources in the files array. It allows browsers or other tooling to verify the integrity of any file you list.
format: Indicates the implemented metadata schema specification. Currently, the default version (Token Metadata JSON Schema V2) is represented by HIP412@2.0.0. If you wonder why, each update to the JSON Schema requires a new Hedera Improvement Proposal (HIP). Therefore, the format field lists the HIP number and the associated version of the Token Metadata JSON Schema.

```
{% endtab %}
{% endtabs %}
```

Putting things together:

```
{% tabs %}
{% tab title="Full schema implementation" %}
This is what a full Token Metadata JSON Schema V2 specification looks like.
```

```
json
{
  "name": "Example NFT 001",
  "creator": "Jane Doe, John Doe",
  "creatorDID":
"did:hedera:mainnet:7Prd74ry1Uct87nZqL3ny7aR7Cg46JamVbJgk8azVgUm;hedera:mainnet:
fid=0.0.123",
  "description": "This describes my NFT",
  "image": "https://myserver.com/preview-image-nft-001.png",
  "checksum":
"ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad",
  "type": "image/png",
  "format": "HIP412@2.0.0",
  "properties": {
    "externalurl": "https://nft.com/mycollection/001"
  },
  "files": [
    {
      "uri": "https://myserver.com/high-resolution-nft-001.png",
      "checksum":
"9defbb6402d4bf39f2ea580099c73194647b24a659b6f6b778e3dd71755b8862",
      "isdefaultfile": true,
      "type": "image/png"
    },
    {
      "uri": "ipfs://yusopwpksaioposjfopiapnnjls1",
      "type": "image/png"
    }
  ],
  "attributes": [
    {
      "traittype": "clothing",
      "value": "pants"
    },
    {
      "traittype": "birth",
      "displaytype": "datetime",

```

```

        "value": 732844800
      },
      "localization": {
        "uri":
"ipfs://QmWS1VAdMD353A6SDk9wNyvkT14kyCiZrNDYAad4w1tKqT/{locale}.json",
        "default": "en",
        "locales": ["es", "fr"]
      }
    }
  }
}

{% endtab %}
{% endtabs %}

```

How do you verify your token metadata is correct?

When you create token metadata for the first time, you want to verify your metadata against the Token Metadata JSON Schema V2. To help you, Hedera has created an NFT utilities SDK (only for JavaScript) to verify your metadata against the JSON Schema V2.

You can install the package using Yarn or NPM.

```

bash
npm i -s @hashgraph/nft-utilities

```

Next, you need to import the validator function that accepts your metadata as a JSON object and the schema version against which you want to verify the metadata (2.0.0). Here's the code.

```

javascript
const metadata = {
  attributes: [
    { traittype: "Background", value: "Yellow" }
  ],
  creator: "NFT artist",
};
const version = '2.0.0';

const issues = validator(metadata, version);
console.log(issues);

```

The package will return errors and warnings using the below interface. This snippet of example output tells you that you have incorrectly used the percentage displaytype in the attributes field, and you have defined a custom field called imagePreview on the root of the metadata object, which is not allowed (use the properties field).

```

json
{
  "errors": [
    {
      "type": "attribute",
      "msg": "Trait stamina of type 'percentage' must be between [0-100],
found 157",
      "path": "instance.attributes[0]"
    }
  ],
  "warnings": [
    {

```

```

        "type": "schema",
        "msg": "is not allowed to have the additional property
'imagePreview'",
        "path": "instance"
    }
]
}

```

Want to learn more about token metadata?

Here's a video about how crucial structuring your token metadata is and how to do it according to Token Metadata JSON Schema V2.

```
{% embed url="https://www.youtube.com/watch?v=o8lY5nQ7pYo" %}
```

You can find examples in this blog post in the section "Token Metadata V2 NFT Examples". If you still have questions, reach out on Discord or ask it on StackOverflow.

Besides that, you can read up on the full implementation of token metadata in the Hedera Improvement Proposals GitHub Repository under HIP-412.

```

<table data-card-size="large" data-view="cards"><thead><tr><th
align="center"></th><th data-hidden data-card-target
data-type="content-ref"></th></tr></thead><tbody><tr><td
align="center"><p>Writer: Michiel, Developer Advocate</p><p><a
href="https://github.com/michielmulders">GitHub</a> | <a
href="https://www.linkedin.com/in/michielmulders/">LinkedIn</a></p></td><td><a
href="https://www.linkedin.com/in/michielmulders/">https://www.linkedin.com/in/
michielmulders/</a></td></tr><tr><td align="center"><p>Editor: Krystal,
Technical Writer</p><p><a href="https://github.com/theekrystallee">GitHub</a> |
<a href="https://twitter.com/theekrystallee">Twitter</a></p></td><td><a
href="https://twitter.com/theekrystallee">https://twitter.com/theekrystallee/</
a></td></tr></tbody></table>

```