# Oracle® Cloud

# Using Oracle Blockchain Platform

F26726-35
March 2024

Oracle Cloud Using Oracle Blockchain Platform,

F26726-35

# Contents

# 7 Build Chaincodes with Low-Code Blockchain App Builder

## 8 Deploy and Manage Chaincodes

## A Node Configuration



## B Using the Fine-Grained Access Control Library Included in the Marbles Sample



## C Run Solidity Smart Contracts with EVM on Oracle Blockchain Platform



## D Updating Applications for Hyperledger Fabric v2.x

# Preface

Learn how to use the service to use and manage blockchains.

**Topics:**



## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

## Related Topics

These related Oracle resources provide more information.

- For a full list of guides, refer to the Books tab in the Oracle Blockchain Platform Help Center.
- Oracle Public Cloud: `http://cloud.oracle.com`
- *Managing and Monitoring Oracle Cloud*

## Conventions

Conventions used in this document are described in this topic.

**Text Conventions**

| Convention | Meaning |
|---|---|
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |

| Convention | Meaning |
| --- | --- |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

### Videos and Images

Your company can use skins and styles to customize the look of the application, dashboards, reports, and other objects. It is possible that the videos and images included in the product documentation look different than the skins and styles your company uses.

Even if your skins and styles are different than those shown in the videos and images, the product behavior and techniques shown and demonstrated are the same.

1

# What's Oracle Blockchain Platform?

This topic contains information to help you understand what Oracle Blockchain Platform is.

**Topics:**



## What's a Blockchain?

A blockchain is a system for maintaining distributed ledgers of facts and the history of the ledger’s updates. A blockchain is a continuously growing list of records, called blocks, that are linked and secured using cryptography.

This allows organizations that don't fully trust each other to agree on the updates submitted to a shared ledger by using peer to peer protocols rather than a central third party or manual offline reconciliation process. Blockchain enables real-time transactions and securely shares tamper-proof data across a trusted business network.

A blockchain network has a founder that creates and maintains the network, and participants that join the network. All organizations included in the network are called members.

Oracle Blockchain Platform is a permissioned blockchain, which provides a closed ecosystem where only invited organizations (or participants) can join the network and keep a copy of the ledger. Permissioned blockchains use an access control layer to enforce which organizations have access to the network. The founding organization, or blockchain network owner, determines the participants that can join the network. All nodes in the network are known and use consensus protocol to ensure that the next block is the only version of truth. There are three steps to consensus protocol:

- **Endorsement —** This step determines whether to accept or reject a transaction.
- **Ordering —** This step sorts all transactions within a time period into a sequence or block.
- **Validation —** This step verifies that the required endorsement are gotten in compliance with the endorsement policy and organization permissions.

**Blockchain's key properties**

**Shared, transparent, and decentralized**— The network maintains a distributed ledger of facts and update history. All network participants see consistent data. Data is distributed and replicated across the network’s organizations. Any authorized organizations can access data.

**Immutable and irreversible** — Each new block contains a reference to the previous block, which creates a chain of data. Data is distributed among the network organizations. Blockchain records can only be appended and can't be undetectably altered or deleted. Consensus is required before blocks or transactions are written to the ledger. Therefore, the existence and validity of a data record can't be denied. After endorsement policies are

satisfied and consensus is reached, data is grouped into blocks and blocks are appended to the ledger with cryptographically secured hashes that provide immutability. Only those members authorized to have the corresponding encryption keys can view data.

**Encryption** — All records are encrypted.

**Closed ecosystem** — Joined organizations can have a copy of the ledger. Organizations are known in the real world. Consensus protocols depend on knowing who the organizations are.

**Speed** — Transactions are verified in minutes. Network members interact directly.

### Blockchain example

An example of an organization that benefits from using blockchain is a supply chain contract manufacturing company. Suppose this company is located in the United States and uses a third-party company in Mexico to source materials for and produce electronic components. With a blockchain network, the manufacturing company can quickly know the answers to the following questions:

- Where is the product in the production cycle?
- Where is the product being produced?
- Does the product contain ethically sourced materials?
- Does the product meet specifications and exporting compliance rules?
- When is ownership transferred?
- Does the invoice match and should the organization pay it?
- How should the organization handle any exceptions to the manufacturing, shipping, or receiving process?

# Why Should I Use Blockchain?

Implementing blockchain can help you manage and bring efficiency to many aspects of your business practices.

The key benefits of using a blockchain are:

**Increase Business Velocity** — You can create a trusted network for business-to-business transactions and extend and automate your operations beyond the enterprise. With blockchain, you can optimize business decisions by providing real-time information visibility across your company's ecosystem.

**Reduce Operation Costs** — Use blockchain to accelerate transactions and eliminate cumbersome offline reconciliations by using a trusted shared fabric of common information. Blockchains help you eliminate intermediaries and related costs, possible single points of failure, and time delay by using a peer to peer business network.

**Reduce the cost of fraud and regulatory compliance** — Blockchain allows you to gain the security of knowing that business critical records are made tamper-proof with securely replicated, cryptographically linked blocks that protect against single point of failure and insider tampering.

# What Are the Advantages of Oracle Blockchain Platform?

Using Oracle Blockchain Platform to create and manage your blockchain network has many advantages over other available blockchain products.

As a preassembled PaaS, Oracle Blockchain Platform includes all the dependencies required to support a blockchain network: compute, storage, containers, identity services, event services, and management services. Oracle Blockchain Platform includes the blockchain network console to support integrated operations. This helps you start developing applications within minutes, and enables you to complete a proof of concept in days or weeks rather than months.

### How Oracle Blockchain Platform Adds Value to Hyperledger Fabric

Oracle Blockchain Platform is based on the Hyperledger Fabric project from the Linux Foundation, and it extends the open source version of Hyperledger Fabric in many ways.

### Provisioning and Integration in Oracle Cloud Infrastructure

- Includes preassembled PaaS with template-based provisioning. See Before You Create Your Instance.
- Uses Oracle Cloud Infrastructure to incorporate infrastructure dependencies (managed containers, virtual machines, identity management, block and object storage).
- Supports multi-cloud, hybrid blockchain network topology that spans multiple Oracle Cloud Infrastructure data centers, on-premises deployments of Hyperledger Fabric, and third-party clouds to link blockchain nodes across organizations, data centers, and continents.

### Operates as an Oracle Managed Service

- Includes Oracle operations monitoring.
- Has zero-downtime managed patching and updates.
- Includes embedded ledger and configuration backups.

### Enhances Security

- Uses data in-transit encryption based on TLS 1.2, prioritizing forward-secrecy ciphers in the TLS cipher-suite.
- Uses data at-rest encryption for all configuration and ledger data.
- Isolates customers from other tenants and the Oracle staff.
- Includes a web application firewall to protect blockchain components against cyberattacks, including predefined Open Web Access Security Project (OWASP) rules, aggregated threat intelligence from multiple sources, and layer 7 distributed denial-of-service (DDoS) attacks.
- Provides audit logging of all API calls to the blockchain resources, with records available through an authenticated, filterable query API or as batched files from Oracle Cloud Infrastructure Object Storage.

### Leverages Built-In Oracle Identity Cloud Service Integration

- Provides user and role management. See Set Up Users and Application Roles.

- Provides authentication for the Oracle Blockchain Platform console, REST Proxy, and CA.
- Supports identity federation and third-party client certificate support to enable consortia formation and simplifies member onboarding.

**Adds REST Proxy**

- Supports a rich set of Fabric APIs through REST calls for simpler transaction integration. See REST API for Oracle Blockchain Platform.
- Enables synchronous and asynchronous invocations. Enables events and callbacks and DevOps operations.
- Simplifies integration and insulates applications from underlying changes in transaction flow.

**Accelerates Integration**

- Provides plug-and-play enterprise adapters using Oracle Integration Cloud Service to integrate Oracle SaaS, PaaS, and on-premises applications with blockchain transactions, queries, and events. See Oracle Integration.
- Blockchain-enabled Oracle Flexcube, Open Banking API Platform, and other Oracle applications with embedded blockchain APIs.
- Enables ERP, EPM, GL, SCM, and HCM business processes in Oracle SaaS, on-premises, and non-Oracle systems to rapidly integrate with blockchain to streamline data exchange and conduct trusted transactions with other organizations.

**Provides the Management and Operations Console**

- Provides a comprehensive, intuitive web user interface and wizards to automate many administration tasks. For example, adding organizations to the network, adding new nodes, creating new channels, deploying chaincodes, browsing the ledger, and more. See Oracle Blockchain documentation library.
- Enables DevOps through REST APIs for administration and monitoring of blockchain.
- Dynamically handles configuration updates without node restart.
- Includes dashboards, ledger browser, and log viewers for monitoring and troubleshooting.

**Replaces Ledger DB World State Store With Oracle Berkeley DB**

- Provides Couch DB rich query support at Level DB performance.
- Provides SQL-based rich query support. See What's the State Database?
- Validates query results at commit time to ensure ledger integrity and avoid phantom reads.

**Integrates Rich History Database**

- Enables transparent shadowing of transaction history and private data collections to Autonomous Data Warehouse or Database as a Service and the use of Analytics or Business Intelligence (for example, Oracle Analytics Cloud or third-party tools) on blockchain transaction history and world state data. See Create the Rich History Database.

- Supports standard tables and blockchain tables for storing rich history. Blockchain tables are tamperproof append-only tables, which can be used as a secure ledger while also being available for transactions and queries with other tables.

**Includes Low-Code Blockchain App Builder**

Blockchain App Builder assists with rapid development, testing, debugging, and deployment of chaincode on Oracle Blockchain Platform networks. Blockchain App Builder generates complex chaincodes in TypeScript (for Node.js chaincode) and Go (for Golang chaincode) from a simple specification file. Blockchain App Builder supports the full development life cycle either from a command-line interface or as an extension for Visual Studio Code.

Blockchain App Builder also includes tokenization support for both fungible and non-fungible tokens. Token classes and methods are automatically generated, and additional token methods are provided so that developers can create complex business logic for tokens.

**Highly Available Architecture and Resilient Infrastructure**

Built for business-critical enterprise applications, Oracle Blockchain Platform is designed for continuous operation as a highly secure, resilient, scalable platform. This platform provides continuous monitoring and autonomous recovery of all network components based on continuous backup of the ledger blocks and configuration information.

Each customer instance uses a framework of multiple managed VMs and containers to ensure high availability. This framework includes:

- Peer node containers distributed across multiple VMs to ensure resiliency if one of the VMs is unavailable or is being patched.
- Orderers, fabric-ca, console, and REST proxy nodes are replicated in all VMs for transparent takeover to avoid outages.
- Isolated VM environments for customer chaincode execution containers for greater security and stability.

Built-in integration with Oracle Identity Cloud Service for user authentication, roles management, and identity federation immediately leverages Oracle Identity Cloud Service accounts and enables easy onboarding of consortium members who prefer using SAML-based federation for authentication against their own identity providers.

Oracle Blockchain Platform is an Oracle managed services in which provisioning, running, and maintaining all of the infrastructure is transparent to customers. The entire framework can be provisioned with only a few clicks and user inputs, such as which shape to use, the initial number of peers, and if the instance type is Founder or Participant. The rest of the instance is automatically defined by the QuickStart shape you selected. See Before You Create an Oracle Blockchain Platform Instance.

The platform is integrated with Oracle Cloud operations management and monitoring service for continuous DevOps. Full stack zero-downtime patching and upgrades are provided with the platform. These are transparently performed by Oracle operations with no customer downtime required. And if any security vulnerabilities are discovered, emergency security patching is enabled for the operating system and all of the components that comprise the service. Ongoing adaptive intelligent cyber-threat detection, mitigation, and remediation are provided as part of the Oracle Cloud Infrastructure security-in-depth approach. This leverages machine learning-based adaptive intelligence for quick detection of intrusions and abnormal behaviors, and automated patching as one of the tools for faster remediation. See Oracle Cloud Infrastructure Documentation.

Oracle Blockchain Platform supported by Oracle Cloud Infrastructure and Oracle Cloud Operations delivers the best-in-class levels of availability, performance, and security. For availability SLAs, see Oracle PaaS and IaaS Public Cloud Services - Pillar Document.

# What Do I Get with Oracle Blockchain Platform?

Your instance includes everything you need to build, run, and monitor a complete production-ready blockchain network based on Hyperledger Fabric.

Your Oracle Blockchain Platform instance is defined by the shape and the underlying platform version of Hyperledger Fabric that you selected when you created your instance. See Before You Create Your Instance. Your instance includes validating peer nodes, a membership services provider (MSP), and an ordering service.

The platform versions for instances, Hyperledger Fabric v2.x and Hyperledger Fabric v1.4.7, use different processes for chaincode lifecycle management. See Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v2.x) or Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v1.4.7).

In addition, REST proxy nodes are provided and a default channel is created. Use the console user interface to further configure, administer, and monitor the network, as well as install, deploy, and upgrade smart contracts (also known as chaincodes). The Developer Tools tab contains sample chaincodes that you can deploy and run to help you quickly understand how the blockchain network works.

Because Oracle Blockchain Platform is part of the Oracle Cloud platform, it's pre-assembled with the underlying cloud services, including containers, compute, storage, identity cloud services for authentication, object store for embedded archiving, and management and log analytics for operations and troubleshooting. You can configure multiple peer nodes and channels for availability, scalability, and confidentiality, and Oracle Cloud will automatically handle the underlying dependencies.

Additional instances can be created for other organizations and joined into your blockchain network. As an Oracle Cloud service, each instance includes replicated resources, monitoring and recovery agents, embedded archiving of configuration data and ledger blocks, and integration with management and log analytics services to help Oracle operations monitor and troubleshoot any issues. It also includes zero-downtime managed patching to resolve any issues and upgrade the service components without interrupting your operations.

2

# Get Started Using Samples

This topic contains information about the samples included in your instance. Using samples is the fastest way for you to get familiar with Oracle Blockchain Platform.

**Topics**



## What Are Chaincode Samples?

Oracle Blockchain Platform includes chaincode samples written in Go and Node.js to help you learn how to implement and manage your network's chaincodes.

To get to the Chaincode Samples page in the Oracle Blockchain Platform console, open the **Developer Tools** tab and select **Samples**.

The Chaincode Samples page contains:

- The Balance Transfer sample is a simple chaincode representing two parties with account balances and operations to query the balances and transfer funds between parties.
- The Marbles sample includes a chaincode to create marbles where each marble has a color and size attribute. You can assign a marble to an owner and enable operations to query status and trade marbles by name or color between owners.
- The Car Dealer sample includes a chaincode to manage the production, transfer, and querying of vehicle parts; the vehicles assembled from these parts; and transfer of the vehicles.
  In this sample, a large auto maker and its dealers and buyers have created a blockchain network to streamline its supply chain activities. Blockchain helps them reduce the time required to reconcile issues with the vehicle and parts audit trail.
- The Fiat Money Token sample includes a chaincode to manage the complete life cycle of a fractional fungible token that represents fiat money. After you initialize the token, create token user accounts, and assign the minter role, you can issue, transfer, and burn tokens. You can also track token account balances and transaction history. For more information about the token samples, see Working With the Sample Token Specification Files.
- The Loyalty Token sample includes a chaincode to manage a loyalty program by using tokens. Loyalty points can be awarded, redeemed, and transferred. For more information about the token samples, see Working With the Sample Token Specification Files.
- The NFT Art Collection Marketplace sample includes a chaincode to simulate a marketplace for buying and selling non-fungible tokens (NFTs) associated with works of art. In this sample, museums can mint (create) NFTs for artworks in the blockchain network. Consumers can then buy and then resell NFTs from the museums. The NFT Art Collection Marketplace sample is designed for chaincode development in TypeScript, and is in available in two versions: one for the ERC-721 token standard, and one for the

ERC-1155 token standard. For more information about the token samples, see Working With the Sample Token Specification Files.

Use the **Download sample here** links under each sample to download the sample chaincode. The download contains the Go and Node.js versions of the chaincode.

The download also contains a Java version of the chaincode.

# Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x)

You can install, deploy, and invoke the sample chaincodes that are included in Oracle Blockchain Platform.

You must be an administrator to install and deploy sample chaincodes. If you've got user permissions, then you can invoke sample chaincodes.

1. Go to the console and select the **Developer Tools** tab.
2. Click the **Samples** pane.

   The Chaincode Samples page is displayed.
3. Locate the sample chaincode and install it.
   a. Choose the sample chaincode that you want to use and click the corresponding **Install** button.
   b. In the Install Chaincode dialog, specify one or more peers to install the chaincode on, and select which chaincode language you want to use (Go, Node.js, or Java). Click **Install**.
4. Deploy the chaincode.
   a. Click the chaincode's **Deploy** button.
   b. In the Deploy Chaincode dialog select the channel you want to deploy the chaincode on. Click **Deploy**.
5. Go to the Channels tab and click the name of the channel that you deployed the sample chaincode on.
   a. In the Channel Information page, click the Deployed Chaincodes pane to confirm the chaincode's deployment on the channel.
   b. You can use the Ledger pane to locate information about individual transactions on the channel.
6. Click the Ledger pane and confirm the following.
   - The Ledger Summary indicates one deployment occurred. A deployment consists of an approval and a commit.
   - In the Ledger table, locate the two blocks with a Type of `data`.
   - Click the first block and in the Transactions table, click the arrow icon to display more information about the block. Confirm that the Function Name field displays `ApproveChaincodeDefinitionForMyOrg`.
   - Click the second block and confirm that the Function Name field displays `CommitChaincodeDefinition`.
7. If needed, go to the Chaincodes tab and deploy the chaincode on other channels.

If you're working on a network that contains multiple members and have deployed the chaincode on the founder, then you don't have to deploy the chaincode on the participants where you installed the same chaincode. In such cases, the chaincode is already deployed on the participants.

a. Locate the package ID of the chaincode you want to deploy in the table and click it.

   The Installed Peers Summary page is displayed.

b. Click **Deployed on Channels**.

c. On the Deployed Channels Summary page, click the **Deploy on a New Channel** button.

d. In the Deploy Chaincode dialog specify the required information, and then click **Deploy**.

8. Invoke the chaincode.

   a. Go to the Chaincode Samples page, locate the chaincode you're working with, and click its **Invoke** button.

   b. In the Invoke Chaincode dialog, select a channel to run the transaction on.

   c. In the **Action** field, specify an action to complete using the chaincode.

   d. Click **Execute**.

9. Confirm whether the chaincode invoked successfully.

   a. Go to the Channels tab, and locate and click the channel the chaincode was installed on.

   b. In the Ledger Summary table, locate the block number that indicates an invocation occurred.

   c. Click the block and confirm that in the Transactions table you see `Success` in the Status column.

10. If needed, go to the Samples page and invoke any other operations on the chaincode.

# Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7)

You can install, instantiate, and invoke the sample chaincodes included in Oracle Blockchain Platform.

Tutorial

You must be an administrator to install and instantiate sample chaincodes. If you've got user permissions, then you can invoke sample chaincodes.

1. Go to the console and select the **Developer Tools** tab.
2. Click the **Samples** pane.

   The Chaincode Samples page is displayed.

3. Locate the sample chaincode and install it.

   a. Choose the sample chaincode that you want to use and click the corresponding **Install** button.

b. In the Install Chaincode dialog, specify one or more peers to install the chaincode on, and select which chaincode language you want to use (Go, Node.js, or Java). Click **Install**.

4. Instantiate the chaincode.
   a. Click the chaincode's **Instantiate** button.
   b. In the Instantiate Chaincode dialog select the channel you want to instantiate the chaincode to, and specify any required parameters. Click **Instantiate**.
5. Go to the Channels tab and click the name of the channel that you instantiated the sample chaincode to.
   a. In the Channel Information page, click the Instantiated Chaincodes pane to confirm the chaincode's deployment on the channel.
   b. You can use the Ledger pane to locate information about individual transactions on the channel.
6. Click the Ledger pane and confirm the following.
   - The Ledger Summary indicates one deployment occurred.
   - In the Ledger table, locate the block with the Type of **data (sys)**.
   - Click the block and in the Transactions table, click the arrow icon to display more information about the block. Confirm that the **Function Name** field displays "deploy."
7. If needed, go to the Chaincodes tab and instantiate the chaincode on other channels.

   If you're working on a network that contains multiple members and have instantiated the chaincode on the founder, then you don't have to instantiate the chaincode on the participants where you installed the same chaincode. In such cases, the chaincode is already instantiated and running on the participants.
   a. Locate the name of the chaincode you want to instantiate in the table and click it.
   b. In the Chaincode Information page, click the **Instantiate on a New Chaincode** button.
   c. In the Instantiate Chaincode dialog specify the required information.
8. Invoke the chaincode.
   a. Go to the Chaincode Samples page, locate the chaincode you're working with, and click its **Invoke** button.
   b. In the Invoke Chaincode dialog, select a channel to run the transaction on.
   c. In the **Action** field, specify an action to execute the chaincode.
   d. Click **Execute**. The Transaction Results shows returned values, and the API details field displays the detailed log of all blockchain processes performed from invoking the transaction.
9. Confirm whether the chaincode invoked successfully.
   a. Go to the Channels tab, and locate and click the channel the chaincode was installed on.
   b. Confirm that the Ledger pane is selected, and in the Query Ledger table, locate the block number indicating that an invocation occurred.

c. Click the block and confirm that in the Transactions table you see “Success” in the **Status** column.

10. If needed, go to the Samples page and invoke any other operations on the chaincode.

3

# Manage the Organization and Network

This topic contains information to help you understand the console and how you can use it to manage the channels and nodes that make up your organization and the blockchain network.

**Topics**



## What's the Console?

The Oracle Blockchain Platform console helps you monitor the blockchain network and perform day to day administrative tasks.

When you provisioned your Oracle Blockchain Platform instance, all of the capabilities you need to begin work on your blockchain network were added to the console.

You can use the console to perform tasks such as managing nodes, configuring network channels and policies, and deploying chaincodes. You can also monitor and troubleshoot the network, view node status, view ledger blocks, and find and view log files.

In most cases, each member of your network has its own console that they use to manage their organization and monitor the blockchain network. Your role in the network (founder or participant) determines the tasks you can perform in your console. For example, if you're a participant, then you can't add another participant to the network. Only the founder can add a participant to the network.

Also, what you can do in the console is determined by your access privileges (either Administrator or User). For example, only an Administrator can set an anchor peer or create a new channel.

Your instance includes sample chaincodes that you can use to get started. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x) or Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7).

The console is divided into tabs.

**Dashboard Tab**

Use the Dashboard tab for an overview of the network's performance. See What Type of Information Is on the Dashboard?

On the Dashboard tab, you'll find:

- A banner showing you how many different components are on your network. For example, how many channels and chaincodes.
- The number of user transactions on a channel during a specific time range.
- The number of nodes that are running or stopped.
- The number of endorsements and commits by peers.
- Utilization statistics for your instance's partitions.

### Network Tab

The Network tab is where you view a list of the members in your network. The first time you use the Network tab after setting up your instance, you'll see the nodes you created during set up.

You can use the Network tab to:

- Find the organization IDs of the members in your network, their Membership Service Provider (MSP) IDs, and roles.
- Add a participant to the network.
- See a graphical representation of the network's structure.
- Configure, view, or import the orderer settings.
- Manage certificates.
- Add new orderering service node into network.
- Export the network config block.

### Nodes Tab

Go to the Nodes tab to view a list of the nodes in your network. The first time you use the Nodes tab after setting up your instance, you'll see:

- The console node.
- The number of peer nodes you requested when provisioning.
- The number of orderer nodes associated with your instance type. Standard has three orderer nodes and cannot be scaled up, while Enterprise has three and additional can be added.
- One Fabric certificate authority (CA) node representing the membership service.
- One REST proxy node.

During the founder instance provisioning a default channel was created and all peers were added to it.

Use the Nodes tab to:

- View and set node configurations.
- Export and import peers.
- Start, stop, and restart nodes.
- Configure and start a new orderer node.
- See a graphical representation of which peer nodes are using which channels.

- Click a node's name to find more information about it.

### Channels Tab

The Channels tab shows you the channels in your network, the peers using the channels, and the chaincodes deployed on the channels. The first time you use the Channels tab after setting up your instance, you'll see the default channel that was created and all of the peers in your network added to it.

Use the Channels tab to:

- Add new channels.
- See the number of chaincodes deployed on a channel.
- Click a channel's name to find more information about it, such as its ledger summary, the peers and OSNs joined to the channel, and the channel's policies and ACLs.
- Join peers to the channel.
- Manage the ordering service of the channel.
- Add or remove an ordering service node (OSN) for a channel.
- View and update the ordering service's settings.
- Configure rich history for the channel.
- Run and analyze rich queries on chaincodes in the channel.
- (Hyperledger Fabric v2.x) Upgrade a chaincode.

### Chaincodes Tab

Note that Oracle Blockchain Platform refers to smart contracts as chaincodes.

Go to the Chaincodes tab to view a list of the chaincode packages installed on the instance. The first time you use the Chaincodes tab after setting up your instance, no chaincodes are displayed in the list because no chaincodes were included during set up. You must add the needed chaincodes.

You can use the Chaincodes tab to:

- Install and deploy a chaincode using the Quick or Advanced deploy option.
- See how many peers have a chaincode installed.
- Find out how many channels a chaincode was deployed on.
- (Hyperledger Fabric v1.4.7) Upgrade a chaincode.

### Developer Tools Tab

The Developer Tools tab is designed to help you learn blockchain fundamentals like how to write chaincodes and create blockchain applications.

You can use the Developer Tools tab to:

- Download Blockchain App Builder for Oracle Blockchain Platform - a set of tools and samples to help you create, test, and debug chaincode projects using a command line interface or a Visual Studio Code extension.
- Find templates and the Hyperledger Fabric mock shim to help you create chaincodes.
- Link to the SDKs and APIs that you need to write blockchain applications.

- Use the sample chaincodes to learn about chaincodes. Install, deploy, and invoke the sample chaincodes.

# Modify the Console Timeout Setting

The Oracle Blockchain Platform console attempts to contact the nodes on the network for 600 seconds before it times out.

In most cases you won't have to adjust this setting, but if the console is frequently not responding, then consider increasing the timeout value. Oracle doesn't recommend decreasing the timeout value.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab go to the nodes table and locate the console node. Use the nodes table's type column to find the Console node.
3. Click the node's **More Actions** menu and then click **Edit Configuration**.

   The Configure dialog is displayed.
4. In the **Request Timeout (s)** field, type or use the arrow buttons to indicate the timeout length in seconds.
5. Click **Submit**.

   The timeout length changes immediately, and you don't have to restart the console.

# Find and Understand Your Oracle Blockchain Platform Version Number

Use this topic to find and understand your Oracle Blockchain Platform instance's version number.

1. Go to the console and in the top right of the screen, locate and click your user name.
2. Select **About**.

   Your instance's version number will look similar to the following example:

   ```
   23.3.3
   ```

   where:

   - 23 is the year
   - 3 is the quarter
   - 3 is the minor release number

# Monitor the Network

The console provides several ways for you to monitor the activity and health of your blockchain network. For example, you can find summary information about the total number of blocks submitted to the ledger, or you can search for and locate information about specific chaincode transactions that happened on a specific channel.

# How Can I Monitor the Blockchain Network?

You can use the console to locate the following sources of information to help you understand what's happening on your network.

**Network Overview Information**

Use the Dashboard tab if you need at-a-glance information about how well the whole network is working and to spot any general issues such as a high rate of failing transactions. See View Network Activity .

**Ledger Summary**

For information about the runtime statistics for transactions on a specific channel, go to the channel's Ledger Summary area. You can drill into a specific transaction for more information about it, such as which member initiated the transaction and which peer endorsed it. See View a Channel's Ledger Activity.

**Node Health**

Use a node's Health Summary area to help you understand how the node is performing on the network; for example, CPU utilization and memory utilization. See:

- View Health Information for a CA Node
- View Health Information for the Console Node
- View Health Information for an Orderer Node
- View Health Information for a Peer Node
- View Health Information for a REST Proxy Node

# What Type of Information Is on the Dashboard?

The console's Dashboard tab provides an overview of how well your network is functioning. You can use this information to identify any issue and to navigate to other tabs in the console where you can learn more about and resolve any issues.

**Summary Bar**

This section shows the components in your network (for example, how many nodes and chaincodes). You can click a component number to go to the console tab for more information or to perform tasks related to the component. If your instance is a development instance, then "Development mode" is displayed in the bottom right of the summary bar.

At the top of the console, you'll see what type of instance you're working with. If you're a network founder, then you'll see "(Founder)". If you're a participant in a network, then the top of your console displays the name of the network you're joined to. For example, "(Participant of <*foundername*>)".

**Health**

This section shows how many nodes are running and how many are stopped in the network. Click the node numbers to go to the Nodes tab to investigate why a node might be stopped, or for more information about the nodes in the network.

The nodes in your network are partitioned inside of a virtual machine (VM). This section also shows the percentage of the partition memory used, and the percentage of CPU and disk used. If the memory percentage is relatively low (for example, 50% or lower), then you can create another peer node without your system's performance decreasing significantly. If the percentage is close to 100, then your system most likely can't support another peer node.

### Channel Activity

This area shows how many blocks have been created and how many transactions have been executed based on the number of blocks created. Note that you might see more blocks created than user transactions. For example, if you create a new channel or you deploy a chaincode, then those are classified as system-level transactions and are included in blocks, but not classified as user transactions. This area shows the top four channels that have handled the most transactions, and for each channel shows the number of transactions that have succeeded and failed.

Note the following information:

- User transactions are transactions that were invoked as part of the chaincode's execution, and not underlying actions such as setting up the network, creating channels, and installing and deploying chaincodes.
- A block can contain multiple user transactions.

You can filter the amount of activity information that is displayed. You can select a set time range (for example, last hour or last week), or you can select **Custom** and pick the dates you want activity information for.

### Peer Activity

This area shows the number of endorsement and commits completed by the network's peer nodes. This area shows the top four peer nodes that have endorsed and committed the most transactions, and for each of those four peers, this area shows the number of endorsements and commits that have succeeded and failed.

Note the following information:

- A transaction is an endorsement, and a commit is when a transaction is written to the block.
- Commits can be either user transactions or system transactions
- Commits are the number of transactions that have been committed to the block. Commits aren't blocks.
- Only specific peers must do endorsements, but all peers must do commits.

You can filter the amount of activity information that is displayed. You can select a set time range (for example, last hour or last week), or you can select **Custom** and pick the dates you want activity information for.

## View Network Activity

Use the console's Dashboard tab to find information about your blockchain network's activities, such as percentage of nodes that are running or stopped, and how successfully the network is executing chaincode transactions.

You can use this information as a starting place and then use the other tabs in the console to drill into any issues that you discover. For information about what displays in the Dashboard tab, see What Type of Information Is on the Dashboard?

1. Go to the console and select the Dashboard tab.
2. To see channel and peer activity information that occurred at a specific time such as for the last week or month, go to the filter dropdown menu and select the time range you want. Select Custom to enter specific begin and end dates and click **Apply**.

# Manage Nodes

This topic contains general information about managing the nodes in your network, such as describing the types of nodes in your blockchain network, how to view your nodes and their topology, how to stop and start them, and how to set logging levels for a node.

## What Types of Nodes Are in a Network?

A blockchain network contain console, peer, orderer, certification authority (CA), and REST proxy nodes. The nodes that display in your console depend upon if you're the founder of or a participant in a network.

For example, if you're a participant in a network, your console won't display an orderer node for that network. If you're a founder, your console displays all node types.

**What nodes are included in a new instance?**

After you provision your instance and access the Nodes tab for the first time, you'll see:

- One console node.
- The number of peers you requested during set up. These peers display with the Peer(Member) type. The maximum number of peer nodes that can be included with an instance is 16.
- An orderer node, or ordering service node (OSN), representing an ordering service.
- A Fabric certificate authority (CA) representing the membership service.
- A REST proxy node.

**I need more information about the different node types**

Use this table to find more information about nodes.

| Node Type | What Does This Node Do? | Displays In Founder or Participant Instance | Number of Nodes per Instance | Can I Add Another Node After Provisioning My Instance? |
|---|---|---|---|---|
| CA | This node provides and manages peer node credentials and member credentials. | Founder<br>Participant | 1 | No |
| Console | This node is the console component. | Founder<br>Participant | 1 | No |
| Orderer | This node provides communication between nodes. It guarantees the delivery of transactions into blocks and blocks into the blockchain.<br>If you're a participant, then you must import the founder's ordering service setting into your instance so that all peer nodes can communicate. | Founder<br>Participant | 3 | Enterprise Edition: Yes<br>Standard Edition: No |
| Peer | This node contains a copy of the ledger and writes transactions to the ledger. This node can also endorse transactions.<br>Your network can contain member or remote peers. | Founder<br>Participant | 2 to 16<br>The number of peer nodes you can add was specified when your instance was created. | Yes |
| REST Proxy | This node maps an application identity to a blockchain member, which allows users and applications to call the Oracle Blockchain Platform REST APIs. | Founder<br>Participant | 1 | No |

# Find Information About Nodes

This section contains information about where in the console you can find information about the nodes in your instance and network.

## View General Information About Nodes

Use the Nodes tab to view general information about all of the nodes in your network. For example, Name, Route, Type, and Status.

You can also use the Nodes tab to drill into details about a specific node. For more information about node types, see What Types of Nodes Are in a Network?

1. Go to the console and select the Nodes tab.

2. In the Nodes tab confirm that the **List View** (and not the **Topology View**) is displaying.

| Column | Description |
|---|---|
| Route | Oracle Blockchain Platform generated the URLs when you provisioned your instance or when you create new nodes.<br>If you use the Hyperledger Fabric SDK, then you need these URLs to specify which peers you want the SDK to interact with. |
| Type | Indicates the node type. |
| MSP ID | Membership Service Provider ID. |
| Status | Indicates if the node is running or down. Also indicates if there's an unapplied configuration change for the node. Note the following statuses:<br>• **Up** — The node is running and working normally.<br>• **Down** — The node is stopped.<br>• **N/A** — This status displays for remote peers because your instance doesn't have the permissions required to get the peer's status. |
| IsConfigured | If the node's configuration was updated you need to restart the node for the updates to take effect. Nodes with the `yes` status are running (and not stopped). |
| More Actions Menu | Your permissions determine the options available from the More Actions menu. If you're an administrator, this button provides links to modify the node's configuration. Administrators and users can stop, start, and restart nodes. |

## Access Information About a Specific Node

Use the Nodes tab to access information about a specific. For example, health information or log files.

1. Go to the console and select the Nodes tab.
2. Click a node's name to go to the Node Information page. The panes that display in the Node Information page depend on the node type you select.

| Pane | Available for Which Node Types? | What can I do in this pane? |
|---|---|---|
| Health | All | View metrics to help you understand how the node is performing on the network. Example of metrics include CPU Utilization and Memory Utilization.<br>For a Peer node, this pane displays information about endorsed and committed transactions. |
| Logs | All | View and download log files to discover and troubleshoot issues with a node. |
| Channels | Peer | View a list of channels the selected peer node is using for its communications with other nodes. Join the peer node to other existing channels as needed. Go to the Channel page to create a new channel and specify which peer nodes can join it. |
| Chaincodes | Peer | View the chaincodes that are installed on the peer node. Go to the Chaincode page to install a new chaincode or upgrade an existing chaincode. |
| Transaction Statistics | REST proxy | View the total queries, failed queries, total invocations, and failed invocations handled by the REST proxy. |

## View a Diagram of the Peers and Channels in the Network

Use the Topology view to access an interactive diagram that shows which network peers are using which channels.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click **Topology View** to see a diagram showing the peer nodes in your network and which channels they're using.
3. Hover over a peer to highlight it and the channels it's using.

## Find Node Configuration Settings

Use the Nodes tab to find a specific node's configuration settings. If you're an administrator, then you can update a node's configuration settings. If you're a user, then you can view a node's configuration settings.

1. Go to the console and select the Nodes tab.
2. Go to the Nodes table, locate the node that you want configuration setting information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed, showing the attributes specific to the node type you selected. See Node Configuration.

# Start and Stop Nodes

You can start or stop CA, orderer, peer, and the REST proxy nodes in your network. You can't start or stop the console node or remote peer nodes.

You can start and stop nodes depending upon the traffic in your network. For example, if network traffic is light, then you can stop unneeded peer nodes and orderer nodes.

You can also restart a node. See Restart a Node.

When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Channel tab and Chaincodes tab. If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. If you stop all peers joined to a channel, then the Channels tab lists the channel, but its information isn't available to view.
Before stopping a node for an extended period of time, you should transfer all this peer's responsibilities to other running peers, and then remove all the responsibilities this peer has.

- Check all other peers' gossip bootstrap address lists, remove the peer address, and add another running peer's address if needed. After peer configuration change, restart the peer.
- Check all channels' anchor peer lists, remove the peer from the anchor peer lists, and add another running peer to the anchor peer list if needed.
- If a channel is joined only to this peer, or if chaincode is deployed only on this peer, you should consider using another running peer to join the same channel and deploy the same chaincode.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to start or stop, and click the node's **More Actions** button.
3. Click either the **Start** or **Stop** option. The node's status changes to either *up* or *down* and information is written to the node's log file.

## Restart a Node

You can restart the CA, orderer, peer, and REST proxy nodes in your network. You can't restart the console node or remote peer nodes.

You should restart a node if it's not responding or running properly, or if you've updated a node's configuration. You can also start or stop a node. See Start and Stop Nodes.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to restart, and click the node's **More Actions** button.
3. Click **Restart**.

   The node's status changes to *restarting* and information is written to the log file.

## Set the Log Level for a Node

If you're an administrator, then you can specify the type of information you want to include in a node's log files. For example, ERROR, WARNING, INFO, or DEBUG.

By default, every node's log level is set to INFO. When developing and testing your network, Oracle suggests that you set the logging level to DEBUG. If you're working in a production environment, then use ERROR.

Only an administrator can change a node's log level setting. If you're a user, then you can view a node's log level settings.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the nodes table, locate the node you want to update, click its **More Actions** menu, and click **Edit Configuration**.

   If you have user permissions, then your console will have the **View** option that you click to see the node's log level setting and other configuration settings.

   The Configure dialog is displayed.
3. In the **Log Level** field, select the log level you want to use.
4. Click **Submit**.

# Manage Channels

This topic contains information about managing the channels in your network, such as how to create and view channels, how to join peers and designate and anchor peer, how to work with policies and access control lists, and how to associate orderers with a channel.

# What Are Channels?

Channels partition and isolate peers and ledger data to provide private and confidential transactions on the blockchain network.

Members define and structure channels to allow specific peers to conduct private and confidential transactions that other members on the same blockchain network can't see or access. Each channel includes:

- Peers
- Shared ledger
- Chaincodes instantiated on the channel
- One or more ordering service nodes
- Channel policy definitions and ACLs where the definitions are applied

Each peer that joins a channel has its own identity that authenticates it to the channel peers and services. Although peers can belong to multiple channels, the information on transactions, ledger state, and channel membership is restricted to peers within each channel.

You can use the Oracle Blockchain Platform console or the Hyperledger Fabric SDK to create channels on your blockchain network. See View Channels.

# View Channels

Members in your network use channels to privately communicate blockchain transactions information.

Use the Channel tab to view a list of the channels in your network, create and monitor channels, specify anchor peers, and upgrade the instantiated chaincodes used on your channels.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.

2. In the channel table, click the channel name you want information about. Note that if all peers joined to the channel are stopped, then the channel is listed but its information isn't available to view.

   The Channel Information page is displayed.

3. Click through the Channel Information page's panes to find information about the channel.

| Section | What can I do in this pane? |
|---|---|
| Ledger | Get information about the channel's ledger activity such as block number and the number of user transactions in the block. Click a block number to drill into information about its transactions. You can use the filter field to specify the summary information that you want to see (for example, information from the last day or last month), or use the custom option to enter start and end times. See View a Channel's Ledger Activity. |

| Section | What can I do in this pane? |
|---|---|
| (Hyperledger Fabric v2.x) Deployed Chaincodes | View the list of chaincodes that have been deployed on the channel. |
| (Hyperledger Fabric v1.4.7) Instantiated Chaincodes | View the list of chaincodes that have been instantiated on the channel. |
| Orderers | View a list of the orderers currently active, and allows you to add a new OSN to join the channel. |
| Peers | View the list of peers that are joined to the channel. Use this section to set anchor peers for the channel. |
| Organizations | View the list of network members whose peers are using the channel to communicate. |
| Channel Policies | View the list of the standard policies and any policies that you created for the channel. Use this section to add, modify, and delete policies. |
| ACLs | View the access control lists (ACLs) and the policies used to manage which organizations and roles can access the channel's resources. |

# Create a Channel

You can add channels to the network and specify which members can use the channel, and which peers can join the channel. You can't delete channels.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
2. In the Channels tab, click **Create a New Channel**.
3. In the **Channel Name** field, enter a unique name for the channel. The channel's name can be up to 128 characters long.
4. In the Organizations section, select any additional members that you want to communicate on the channel.

   If you're working in a participant instance, you need to add the founder to your instance before the founder's MSP ID displays in the Organization section. To add the founder organization, go to the Network tab and click the **Add Organization** button to upload the founder's certificates.
5. In the **MSP ID ACL** section, specify the organizations that have access to the channel and permissions for each selected organization. Note that you can add more organizations to or delete them from the channel later, as needed.

   Your organization's permissions are set to write (ReaderWriter) and you can't modify this setting. By default, other member's permissions are set to write (ReaderWriter), but you can change them to read (ReaderOnly) if you don't want the members to invoke chaincodes and to only read channel information and blocks on the channel.
6. (Optional) In the **Peers to Join Channel** field, select one or more peers. Note the following information:

   - If your network contains participants, the participants' peers don't display in this list. Participants must use their consoles to join peers to the channel. A participant can't join its peers to the channel unless its organization was added to the channel's MSP ID ACL section.

- If you want to create the channel only, then don't select any peers. You can add peers to the channel later.

7. Click **Submit**.

   The channel table displays the new channel.

After you create the channel, you can:

- (Hyperledger Fabric v2.x) Deploy a chaincode on the channel. See Deploy a Chaincode.
- (Hyperledger Fabric v1.4.7) Instantiate a chaincode on the channel. See Instantiate a Chaincode.
- If the network contains participants, then they use their consoles to join member peers to the channel. See Join a Peer to a Channel.

## View a Channel's Ledger Activity

Use the ledger to find summary information and runtime statistics for transactions on a specific channel.

1. Go to the console and select the Channels tab.
2. In the channel table, click the channel name that you want transaction information about. In the Channel Information page, confirm that the Ledger pane is selected.
3. Use the Ledger Summary area to find basic information about the channel's activity, such as the total number of blocks in the ledger's chain and the total number of user transactions on the channel.
4. To see blockchain activity that occurred at a specific time such as the last day or week, use the filter drop-down list to select the time range that you want. To locate and drill down into a specific set of transactions, select **Custom** and enter search criteria in the **Start Time** and **End Time** fields, or click the calendar icon and pick the dates that you want. Click **Apply**.

   If you select a specific time period (for example, **Last day**) and then select it again to re-run the query, the query doesn't run again. To get the latest information, click the **Refresh** button.

   The following transaction types can be displayed for a block:

   - genesis — The transaction that runs the configuration block to initialize the channel.
   - data (sys) — The transaction that starts the chaincode's container to make the chaincode available for use.
   - data — A chaincode transaction called for execution on the channel.

5. To find more information about a specific transaction, locate the transaction in the query ledger table and click it. The transactions table displays the transaction's details.

   For any given block, the transactions in the table are listed in the order of the transaction number, which is assigned by the ordering service when the block is created. Because of this, the transactions listed in the table might have time stamps (which are from the peer's endorsement of the chaincode) that are before or after other transactions in the same block. The time range of transactions in a single block is governed by ordering service settings including the batch timeout

parameter (the time that the ordering service waits for additional transactions after an initial transaction before cutting a block).

| Transaction Detail | Description |
|---|---|
| TxID | The unique alphanumeric ID assigned to the transaction. The TxID is constructed as a hash of a nonce concatenated with the signing identity's serialized bytes. |
| Time | The transaction's time stamp (date and time that the transaction occurred). |
| Chaincode | The name of the chaincode that executed the transaction. This field can show the name of a chaincode that you wrote, installed, and deployed, but can also show a system chaincode.<br>System chaincode options are:<br>• (Hyperledger Fabric v2.x) _lifecycle — For lifecycle requests, such as install, deploy, and upgrade.<br>• (Hyperledger Fabric v1.4.7) LSCC — For lifecycle requests, such as instantiate, install, and upgrade.<br>• QSCC — For querying. This chaincode includes APIs for ledger query. |
| Status | Status that indicates whether the transaction succeeded or failed. |

6. Click the triangle icon next to the TxID to view in-depth information about the transaction, such as function name, arguments, validation results, response status, the initiator, and the endorser.

   If a transaction failed, you can use the TxID to search the error logs on the peer node or orderer nodes for more information.

## View or Update a Channel's Organizations List

You can view the list of the organizations that have access to the channel. If you created the channel, then you can change an organization's permissions on the channel, and you can add organizations to or remove them from the channel

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels in your network.

2. In the channels table, locate the channel that you want information about, click the channels **More Actions** button, and click **Edit Channel Organizations**.

   The Edit Organizations page is displayed.

3. In the **MSP ID ACL** section, you can do the following:

   - Modify an organization's permissions. The organization that created the channel is set to write (ReaderWriter). You can't change this setting.
   - If you're the network founder, then clear an organization's checkbox to delete it from the channel. If you're a network participant, then use the **Delete** button to delete an organization from the channel. If you delete an organization from a channel, then the organization and its peers can no longer query, invoke, and instantiate a chaincode on the channel. And the removed organization's peers can't join the channel.
   - Click an organization's checkbox to add the organization to the channel and set its permissions. By default, each member's permissions is set to write (ReaderWriter),

but you can change it to read (ReaderOnly) if you don't want the member to invoke chaincodes and to only read channel information and blocks on the channel.

4. Click **Submit** to save the changes.

## Join a Peer to a Channel

You can add a peer node to a channel so that the node can use it to exchange private transaction information with other peer nodes on the channel.

Note the following information:

- When you create a channel, you specify which local peer nodes can join the channel.
- If you're creating a network containing a participant, then you can select the participant as a member on the channel. Or you can add the participant after the channel is created.
- Your instance has multiple availability domains or fault domains, and Oracle recommends that you join one peer from each partition to the channel. This is because if one VM is unavailable that the channel is still available for endorsements and commits. To determine which domain a peer is located in, in the **More Actions** menu select **Show AD Info** to see the availability domain information.
- You can join a maximum of seven peers from each domain.

See Create a Channel.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the peer node that you want to add to a channel.
3. In the Node Information page, click the Channels pane to view the list of channels the peer is already using.
4. Click **Join New Channels**.

   The Join New Channels dialog is displayed.
5. Click the **Channel Name** field and from the list select the name of the channel to join. Click the field again to select another channel. Click **Join**.

## Add an Anchor Peer

Each member using a channel must designate at least one anchor peer. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

You can designate one or more peers in your organization as an anchor peer on a channel. For a high availability network, you can specify two or more anchor peers. All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

The Channels tab is displayed and the channel table contains a list of all of the channels on your network.

2. In the channels table, click the channel name you want to add anchor peers to.

   The Channel Information page is displayed.

3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to designate as anchor peers and click their **Anchor Peer** checkboxes to select them.
5. Click the **Apply** button.

## Change or Remove an Anchor Peer

You can change or remove a channel's anchor peers. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

Before you change or remove the channel's anchor peers, note the following information:

- To communicate on the channel, you must designate one or more peers in your organization as an anchor peer.
- For a high availability network, you can specify two or more anchor peers.
- All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.

2. In the channels table, click the channel name you want to remove anchor peers from.

   The Channel Information page is displayed.

3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to remove as anchor peers and clear their **Anchor Peer** checkboxes. Alternatively, to add another peer as an anchor peer, click its **Anchor Peer** checkbox to select it.
5. Click the **Apply** button.

## View Information About Deployed Chaincodes

You can view information about the chaincodes that are deployed on the different channels in your network.

You might need information about deployed chaincodes to determine if you need to upgrade the chaincode, or to find out which channels the chaincode was deployed on.

1. Go to the console and select the Channels tab.
2. In the channels table, click the channel name with the chaincode that you want to view information for.
3. In the Channel Information page, confirm that the Deployed Chaincodes pane is selected

4. In the chaincode table, you can:
   - Click the chaincode package ID to go to the Chaincodes tab to learn more information about it. For example, the peers that the chaincode is installed on and the channels that the chaincode is deployed on.
   - In a chaincode's More Actions menu, click **View Chaincode Definition** to find details about the chaincode's definition, including the endorsement policy.
5. (Optional) If you see a channel listing without a chaincode, then you can go to the Chaincodes tab and deploy a chaincode to the channel. See Deploy a Chaincode.

# Work With Channel Policies and ACLs

This topic contains information about a channel's policies and access control lists (ACLs). It provides an overview of what policies are, policy types, and how to modify them, as well as how to use ACLs to manage which organizations and roles can access a channel's resources.

## What Are Channel Policies? (Hyperledger Fabric v2.x)

A policy defines a set of conditions. The required parties must meet the policy's conditions before their signatures are considered valid and the corresponding request happens on the network.

The blockchain network is managed by these policies. Policies check the identity associated with a request against the policy associated with the resource needed to fulfill the request. Policies are located in the channel's configuration.

After you configure the channel's policies, you assign them to the channel's ACLs resources to determine which members are required to sign before a change or action can happen on the channel. For example, suppose you modified the Writers policy to include members from Organization A or Organization B. Then you assigned the Writers policy to the channel's `cscc/GetConfigBlock` ACL resource. Now only a member from Organization A or Organization B can call `GetConfigBlock` on the `cscc` component.

**What Are the Policy Types?**

There are two policy types: Signature and ImplicitMeta.

- **Signature** — Specifies a combination of evaluation rules. It supports combinations of *AND*, *OR*, and *NOutOf*. For example, you could define something like "An admin of org A and 2 other admins" or "11 of 20 org admins."
  Any new policies you create will be Signature policies.
- **ImplicitMeta** — This policy type is only valid in the context of configuration. It aggregates the result of evaluating policies deeper in the configuration hierarchy, which are defined by Signature policies. It supports default rules, for example "A majority of the organization admin policies."

**When Are Policies Created?**

When you add a channel to the network, Oracle Blockchain Platform creates default policies. The default policies are: Admins, Writers, Readers, Endorsement, LifecycleEndorsement (ImplicitMeta policies), and Creator (Signature policy). If needed, you can modify these policies or create new policies.

Note the following important issue about channel policies:

- You can use the console to create a channel and set your organization's ACL to ReaderOnly. After you save the new channel, you can't update this ACL setting from the channel's Edit Organization option.

  However, you can use the console's Manage Channel Policies functionality to add your organization to the Writers policy, which overwrites the channel's ReaderOnly ACL setting.

## What Are Channel Policies? (Hyperledger Fabric v1.4.7)

A policy defines a set of conditions. The required parties must meet the policy's conditions before their signatures are considered valid and the corresponding request happens on the network.

The blockchain network is managed by these policies. Policies check the identity associated with a request against the policy associated with the resource needed to fulfill the request. Policies are located in the channel's configuration.

After you configure the channel's policies, you assign them to the channel's ACLs resources to determine which members are required to sign before a change or action can happen on the channel. For example, suppose you modified the Writers policy to include members from Organization A or Organization B. Then you assigned the Writers policy to the channel's cscc/GetConfigBlock ACL resource. Now only a member from Organization A or Organization B can call GetConfigBlock on the cscc component.

**What Are the Policy Types?**

There are two policy types: Signature and ImplicitMeta.

- **Signature** — Specifies a combination of evaluation rules. It supports combinations of *AND*, *OR*, and *NOutOf*. For example, you could define something like “An admin of org A and 2 other admins" or "11 of 20 org admins.”
  Note that when you modify the Oracle Blockchain Platform's default Admins policy, which was created as an ImplicitMeta policy, you'll use the Signature policy. Any new policies you create will be Signature policies.
- **ImplicitMeta** — This policy type is only valid in the context of configuration. It aggregates the result of evaluating policies deeper in the configuration hierarchy, which are defined by Signature policies. It supports default rules, for example “A majority of the organization admin policies.”
  Oracle Blockchain Platform uses the ImplicitMeta policy type to create the Admins policy. When you modify the Admins policy, you'll use the Signature policy. You can't create or modify any policies using the ImplicitMeta policy. Oracle Blockchain Platform only supports modifying or creating policies using the Signature policy type.

**When Are Policies Created?**

When you add a channel to the network, Oracle Blockchain Platform creates default policies. The default policies are: Admins (ImplicitMeta policy), Creator, Writers, and Readers (Signature policies). If needed, you can modify these policies or create new policies.

Note the following important issues about channel policies:

- You can use the console to create a channel and set your organization's ACL to ReaderOnly. After you save the new channel, you can't update this ACL setting from the channel's Edit Organization option.

However, you can use the console's Manage Channel Policies functionality to add your organization to the Writers policy, which overwrites the channel's ReaderOnly ACL setting.

- When you use the Hyperledger Fabric SDKs to create a channel, Fabric uses the ImplicitMeta policies as the default channel policies for Readers and Writers. When the channel uses these policies, the Oracle Blockchain Platform console can't guarantee that the administrative operations (for example, edit organization) will be successfully processed.

  To correct this issue, update the readers and writers policies to Signature policies, and define the policy rules as needed. See https://hyperledger-fabric.readthedocs.io/en/release-1.3/access_control.html
- When you use the Hyperledger Fabric SDKs or CLI to create a channel, the Creator policy isn't included in the configtx.yaml file. The Creator policy is required by Oracle Blockchain Platform to allow the channel creator to edit a channel's configuration. You must manually edit the configtx.yaml file and add the Creator policy.

## Add or Modify a Channel's Policies

You can add or modify a channel's policy to specify which members are required to perform a specific action on the channel. After you define policies, you assign them to the channel's ACLs.

Before you add or update policies, you need to understand how Oracle Blockchain Platform creates default channel policies. See What Are Channel Policies? (Hyperledger Fabric v1.4.7) or What Are Channel Policies? (Hyperledger Fabric v2.x).

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name that you want to add policies to or modify policies for.
   The Channel Information page is displayed.
3. In the Channel Information page, click the Channel Policies pane.
4. Do one of the following:
   - To add a new policy, click the **Create a New Policy** button. The **Create Policy** dialog is displayed. Enter a name in the **Policy Name** field and select Signature in the **Policy Type** field. Expand the **Signature Policy** section.
   - To modify an existing policy, click a policy's name. The **Update Policy** dialog is displayed.
5. Click the **Add Identity** button to add an organization. Or modify an existing signature policy as needed. Note the following information:

| Field | Description |
|---|---|
| MSP ID | From the dropdown menu, select the organization that must sign the policy. |

| Field | Description |
|---|---|
| Role | Select the corresponding peer role required by the policy. Usually this will be member. You can find a peer's role by viewing its configuration information. |
| Policy Expression Mode | In most cases, you'll use **Basic**. Select **Advanced** to write an expression string using *AND*, *OR*, and *NOutOf*. For information about how to write a valid policy expression string, see Endorsement policies in the Hyperledger Fabric documentation. |
| Signed By | Select how many members must sign the policy to fulfill the request. |

6. If you're adding a new policy, then click **Create**. If you're modifying a policy, then click **Update**.

## Delete a Channel's Policies

You can delete channel policies that you have created.

You can't delete the default policies: Admins, Creator, Readers, Writers, Endorsement, and LifecycleEndorsement. Also, you can't delete a channel policy if it is assigned to an ACL. Before you try to delete a channel policy, confirm that the policy isn't assigned.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel that you want to delete a policy from.
   The Channel Information page is displayed.
3. In the Channel Information page, click the Channel Policies pane.
4. Locate the policy that you want to delete and click its **More Options** button.
5. Click **Remove** and confirm the deletion.

## What Are Channel ACLs?

Access control lists (ACLs) use policies to manage which organizations and roles can access a channel's resources.

Users interact with the blockchain network by targeting components such as the query system chaincode (`qscc`), lifecycle system chaincode (`_lifecycle` on Hyperledger Fabric v2.x, `lscc` on Hyperledger Fabric v1.4.7), configuration system chaincode (`cscc`), peer, and event. These components are associated with specific resources (for example, `GetConfigBlock` or `GetChaincodeData`) that you can assign policies to at the channel level. These policies are a part of the channel's configuration.

A policy defines which organizations and roles can request a resource. When a request is made, the policy tells the system to check the requester's identity and determine if it's authorized to make the request. When you create a channel, Oracle Blockchain Platform includes the default Hyperledger Fabric ACLs with the channel. Oracle Blockchain Platform also creates default policies (Admin, Creator, Writers, and Readers; also Endorsement and LifecycleEndorsement on Hyperledger Fabric v2.x) for the channel. You can modify these

policies or create new policies as needed. See What Are Channel Policies? (Hyperledger Fabric v2.x) or What Are Channel Policies? (Hyperledger Fabric v1.4.7).

## Update Channel ACLs

You can update the channel's ACLs by assigning policies to the channel's resources. A policy defines which organizations and roles can request a resource

Before you update a channel's ACLs, you should understand what policies and ACLs are. See What Are Channel Policies? (Hyperledger Fabric v1.4.7) and What Are Channel ACLs?

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the name of the channel that you want to update ACLs for.
   The Channel Information page is displayed.
3. In the Channel Information page, click the ACLs pane.
4. In the Resources table, locate the resource that you want to update. Click the resource's **Expand** button and select the policy that you want to assign to the resource.
5. Modify the other resource's policies as needed.
6. Click **Update ACLs**.

# Add or Remove Orderers To or From a Channel

The orderer admin organization can add or remove orderers from a channel.

To add orderers to a channel:

1. In the founder console, open the Channels tab and select the channel to see its details view.
2. Open the Orderers subtab. All orderer nodes currently joined to the channel are listed.
3. Click **Join Channel**. Select an OSN not yet in this channel and click **Join**.

To remove orderers from a channel:

1. In the founder console, open the Channels tab and select the channel to see its details view.
2. Open the Orderers subtab. All orderer nodes currently joined to the channel are listed.
3. Select the orderer you want to remove from the channel and from its More Actions menu select **Remove**.

# Set the Orderer Administrator Organization

You can assign the administration of OSNs in a channel to any organization. Normally either the founder or the channel creator would be assigned.

1. In the founder console, open the Channels tab.

2. Select the channel for which you want to set the orderer administrator organization, and from the Action menu select **Manage OSNs Admin**.
3. Select from the list of available organizations, and click **Submit**.

# Edit Ordering Service Settings for a Channel

You can update the ordering service settings for a particular channel.

Note the following important information about editing the ordering service settings for a channel:

- Separately you can update the ordering service settings for the entire network as described in Edit Ordering Service Settings for the Network.
- If you change the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.
- It isn't common, but in some situations, you might expose a different ordering service to some of the network participants. In this case, you'll export the updated network config block and the required participants will import the revised settings. See Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

You must be an administrator to perform this task.

1. Go to the founder's console and select the Channels tab.
2. Locate the channel, click the **More Actions** menu, and select **Update Ordering Service Settings**.

   The Ordering Service Settings dialog is displayed.
3. Update the settings as needed.

| Field | Description |
|---|---|
| Batch Timout (ms) | Specify the amount of time in milliseconds that the system should wait before creating a batch. Enter a number between 1 and 3600000. |
| Max Message Count | Specify the maximum number of message to include in a batch. Enter a number between 1 and 4294967295. |
| Absolute Message Bytes | Specify the maximum number of bytes allowed for the serialized messages in a batch.<br>This number must be larger than the value you enter in the Preferred Message Bytes field. |
| Preferred Message Bytes | Specify the preferred number of bytes allowed for the serialized messages in a batch. A message larger than this size results in a larger batch, but the batch size will be equal to or less than the number of bytes you specified in the Absolute Message Bytes field.<br>Oracle recommends that you set this value to 1 MB or less.<br>The value that you enter in this field must be smaller than the value you enter in the Absolute Message Bytes field. |
| Snapshot Interval Size | Defines number of MB per which a snapshot is taken. |

4. Click **Update**.

   The updated settings are saved.

# Manage Certificates

This topic contains information about how to manage your network's certificates, including how to import and export certificates to set up your blockchain network, and how to manage and revoke certificates.

## Typical Workflows to Manage Certificates

Here are the common tasks for managing your network's certificates.

### Adding Organizations to the Network

You must be an administrator to perform these tasks.

| Task | Description | More Information |
|---|---|---|
| Export or prepare an organization's certificates | The organization that wants to join the network either outputs or writes its certificates file and gives it to the founder. | Export Certificates<br>Create a Fabric Organization's Certificates File<br>Create an Organization's Third-Party Certificates File |
| Import member certificates | The founder imports the organization's certificates file to add the organization to the network. | Import Certificates to Add Organizations to the Network |
| View certificates | The founder can view and manage the network's certificates. | View and Manage Certificates |

### Revoking Certificates

You must be an administrator to perform these tasks.

| Task | Description | More Information |
|---|---|---|
| Decide which certificates to revoke | View the certificates on your system to determine which ones to revoke to keep the network secure. | View and Manage Certificates |
| Select the certificates to revoke | Revoke the certificates in your CA. | Revoke Certificates |
| Apply CRL | Generates and applies an updated CRL to ensure that clients with revoked certificates can't access channels. | Apply the CRL |

# Export Certificates

Founders and participant organizations must import and export certificate JSON files to create the network.

Note the following information:

- For the founder to add a participant organization to the blockchain network, the participant must export its certificates file and make it available to the founder. The founder then uploads the certificates file to add the participant organization to the network.
- The certificate export file contains admincerts, cacerts, and tlscacerts.
- You might need to export certificates for blockchain or application developers. For example, a client application needs the TLS certificate to interact with peers or orderers.

For information about writing certificate files required to add Hyperledger Fabric or Third-Party organizations to the network, see Extend the Network.

1. Go to the console and select the Network tab.
2. In the Network tab, go to the Organizations table, locate the member that you want to export certificates for, and click its **More Actions** button.
3. Click **Export Certificates**.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. Specify where to save the file. Click **OK** to save the certificates file.
5. Send the certificates JSON file to the founder for import. See Import Certificates to Add Organizations to the Network.

# Import Certificates to Add Organizations to the Network

To add an organization to the network, the founder must import a certificates file that was exported or prepared by the organization that wants to join the network.

You can import certificates for the following organization types.

| Type | Description |
|---|---|
| Oracle Blockchain Platform Participant Organization | You can import a participant organization into a Oracle Blockchain Platform network. You upload the certificates that the participant organization exported from the console and sent to you.<br>For information about creating certificates for upload and a list of the other steps that you need to perform to successfully set up a participant organization on the network, see Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service. |

| Type | Description |
|---|---|
| Hyperledger Fabric Organization | You can import a Hyperledger Fabric organization into an Oracle Blockchain Platform network. To successfully upload a Fabric organization's certificates file, you must modify the certificates file to replace all instances of \n with the newline character.<br>See Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network. |
| Third-Party Certificate Organization | You can import an organization that is using certificates generated from a third-party CA server. To successfully upload a third-party organization's certificates file, you must modify the certificates file to replace all instances of \n with the newline character.<br>See Typical Workflow to Join an Organization With Third-Party Certificates to an Oracle Blockchain Platform Network. |

You must be an administrator to import certificates.

1. Go to the console and select the Network tab.
2. In the Network tab, click **Add Organizations**. The Add Organizations page is displayed.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
3. Click **Upload Organization Certificates**. The File Upload dialog is displayed.
4. Browse for and select the JSON file containing the certificate information for the organization you want to add to the network. Usually this file is named `certs.json`. Click **Open**.
5. (Optional) Click the plus (+) icon to locate and upload another organization's certificate information.
6. Click **Add**. The organizations that you added are displayed in the Organization table.

   Note the following information for Oracle Blockchain Platform participant, Hyperledger Fabric, and third-party certificate organizations. Even though the founder uploaded the certificate information, the added organization can't use the ordering service to communicate on the network until it imports the founder's ordering service settings. The founder must export its ordering service settings and give the resulting file to the joining organizations for import. See one of the following:

   - For Oracle Blockchain Platform participants, see Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.
   - For Hyperledger Fabric organizations, see Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network.
   - For third-party certificate organizations, see Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network.

## What's a Certificate Revocation List?

You use a certificate revocation list (CRL) to help manage the certificates throughout your network.

A CRL is a list of digital certificates that the issuing Certificate Authority (CA) has revoked before their scheduled expiration date and should no longer be trusted and used on the network. For example, you should revoke any certificates that have been lost, stolen, or compromised.

After you use the Manage Certificates functionality to revoke certificates for users, Oracle Blockchain Platform creates the CRL. To ensure that the certificates are revoked throughout the network, you'll need to:

- Use the Apply CRL functionality after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

## View and Manage Certificates

Use the console to view and manage the user certificates in your instance and any of the certificates you imported when building the network.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization's ID and click its **More Actions** button. Select **Manage Client Certificates**.

   Note that the Certificate Summary table will be empty until you add users to your instance. Also, the administrator's certificate doesn't display in this table. This is to prevent you from accidentally revoking the administrator's certificate.

   Organizations with third-party certificates or Hyperledger Fabric organization with revoked certificates won't display in this table. In such cases, you must use the native Hyperledger Fabric CLI or SDK to import the organization's certificate revocation list (CRL) file.

   The Certificates Summary dialog is displayed and shows a list of the certificates in your instance.
3. As needed, perform any of the following tasks:
   - Revoke certificates. See Revoke Certificates.
   - If you've revoked certificates and are working in a network with multiple members, then use Apply CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

## Revoke Certificates

An organization can revoke certificates for any of its users. To ensure that the network remains secure, you should revoke certificates in case they're lost, stolen, or compromised.

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.

2. In the Network tab, locate your organization's ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.

3. In the Certificates Summary dialog, locate and select the IDs of the users that you want to revoke certificates for.
4. Click Revoke and confirm that you want to permanently revoke certificates for the selected users.

   The users with revoked certificate display in the table and are added to the CRL.

5. If you're working in a network with other members, then to ensure that their revoked certificates are cleaned up across the network, you must do the following:

   - If you're working in a network with multiple members, then apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

## Apply the CRL

If you're working in a network, then you must apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents members with revoked certificates from accessing the channel.

You must do the following tasks before applying the CRL:

- Revoke certificates. See Revoke Certificates

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization's ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.

3. Click the Apply CRL button and confirm that you want to apply the CRL.

# Manage Ordering Service

This topic contains information about how founders and participants manage the ordering service.

In addition to the content covered in this topic, several channel-specific tasks for the orderer nodes can be performed on the Channels page of the console. See:

- Add or Remove Orderers To or From a Channel
- Set the Orderer Administrator Organization
- Edit Ordering Service Settings for a Channel

## What is the Ordering Service?

Oracle Blockchain Platform supports Raft as the consensus type.

For more information on the Hyperledger Fabric implementation of the Raft protocol, see: The Ordering Service - Raft.

With the older Kafka consensus type, the whole network can have at most two orderer nodes, and they have to join all channels. In some cases, they may be overloaded, and cannot be scaled out. With the Raft consensus type, the network can have an arbitrary number of orderer nodes, and each channel can define its own orderer node set. Different channels can use different orderer nodes, and orderer nodes will no longer be the bottleneck.

However, the Raft consensus type can be complicated to configure properly. There are rules about what can or can't be done, and if these rules are not followed the channel and even the network may not work. The following guidelines should reduce the problems you encounter:

#### Keep the Majority of the Ordering Service Nodes (OSN) Alive

The Raft consensus algorithm requires that the majority of ordering service nodes (OSNs) are working; otherwise no consensus can be made. Majority means greater than 50%. For example, for five OSNs, there must be at least three OSNs working; for six OSNs, there must be at least four OSNs working.

- If there are 50% or less OSNs working in the network, network management will no longer be functional. No new channels can be created, no new orderer nodes can be added into network, no orderer can be removed from network, and so on.
- If there are 50% or less OSNs working in the application channel, no transaction can be submitted to this application channel. Queries may still function correctly, however administrative operations such as adding a new organization, changing the access control list, or instantiating or deploying chaincodes will fail.

Be cautious when adding a new OSN to the network or an application channel. Ensure the owner is trustworthy and the OSN is robust.

When removing OSNs or an organization, ensure that more than 50% of the OSNs will remain working. For example, if you had 2 organizations with 3 OSNs each, if you removed one organization, during the removal it would be interpreted as only 50% of the OSNs being functional. Add an OSN to the remaining organization before deleting the extraneous organization to ensure that you always exceed 50% of the OSNs working.

#### Do Not Add or Remove Orderers Frequently

Every time a new OSN is added into a network or channel, or an existing OSN is removed from a channel, the current Raft OSN cluster will briefly become unstable. During this period, no transactions can be handled, and an error message similar to the following may indicate such a status:

```
UNKNOWN: Stream removed
SERVICE UNAVAILABLE
BAD REQUEST
```

This may last a few minutes. If you have removed the previous Raft leader OSN from the channel, this may last as long as 20 minutes.

Ensure that you aren't adding or removing orderers frequently. If multiple orderers must be added or removed, do one at a time ensuring that the network has returned to operational status before making the next change.

#### Ensure the New Orderer is Started As Soon As Possible

When adding a new orderer into network, usually two organizations will be involved: the founder and the owner of the new orderer. Both parties must follow the instructions in Join the

Participant or Scaled-Out OSNs to the Founder's Ordering Service all the way to completion or the founder won't be able to manage the network.

# Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service

When you provision a participant instance, it is created with 3 orderers. There orderers are inactive until they are joined to a network. When you scale out a founder, the new orderers are also inactive until they are joined to a network.

If multiple orderers must be added or removed, do one at a time ensuring that the network has returned to operational status before making the next change. See What is the Ordering Service? for additional important details about adding, removing, starting, and stopping Raft orderers.

### Export the OSN Settings From the Participant or Scaled-Out Orderers

To join the participant or scaled-out orderers to a network, you need to export their settings and import them into the founder.

1. In the participant console (or the founder console for scaled-out orderers), on the Node tab find the orderer node (or the first orderer node if multiple nodes exist). Select the Action menu for this node and select **Export OSN Settings**.

   This will generate a JSON file with the settings and save the file. The file contains the organization's certificate and the selected orderer service node (OSN) settings signed by the private key of the administrator of the participant organization. This file needs to be sent to the administrator of the founder instance.

   Applications being run on channels using this OSN also require this exported TLS certificate. See Before You Develop an Application.

2. In the founder console, open the Network tab. Click **Add OSN**. A window opens prompting you for the location of the JSON file provided by the participant. Select to upload the file and click **Add**.

   The participant organization or newly scaled-out orderer will be added to the orderer organization section of the system channel list.

### Export the Founder's Configuration Settings

Once the participant or scaled-out orderers have been added to the founder, you need to export the founder's settings and import them to the participant or scaled-out orderer.

1. In the founder console, open the Network tab. Click **Export Network Config Block**.

   The network configuration block contains the latest system channel configuration block. This can be saved and sent to the participant administrator.

2. In the participant console (or the founder console for scaled-out orderers), on the Node tab find the orderer node (or the first orderer node if multiple nodes exist). Select the Action menu for this node and select **Import Network Config Block**.

   You'll be prompted for the file sent by the founder instance administrator.

3. In the participant console, refresh the Node tab. The orderer node status should be listed as "down". From the Action menu select **Start**.

Each orderer node started will be added to the Raft cluster in the founder.

Each time a new OSN is added by scaling out the orderer (as described in Scale Your Instance) these steps need to be repeated to add the new OSN to the Raft cluster.

> **Note:**
>
> You can't add multiple OSNs into a network in a single batch. Ensure only 1 OSN is added at a time.

## Edit Ordering Service Settings for the Network

You can update the ordering service settings for the founder instance.

Note the following important information about editing the ordering service settings:

- The updated settings are used when you create new channels and are not applied to existing channels.
- Separately you can update the ordering service settings for an individual existing channels as described in Edit Ordering Service Settings for a Channel.
- If you change the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.
- It isn't common, but in some situations, you might expose a different ordering service to some of the network participants. In this case, you'll export the updated network config block and the required participants will import the revised settings. See Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

You must be an administrator to perform this task.

1. Go to the founder's console and select the Network tab.
2. Click the **Ordering Service Settings** button.

   The Ordering Service Settings dialog is displayed.
3. Update the settings as needed.

| Field | Description |
|---|---|
| Batch Timeout (ms) | Specify the amount of time in milliseconds that the system should wait before creating a batch. Enter a number between 1 and 3600000. |
| Max Message Count | Specify the maximum number of message to include in a batch. Enter a number between 1 and 4294967295. |
| Absolute Message Bytes | Specify the maximum number of bytes allowed for the serialized messages in a batch.<br>This number must be larger than the value you enter in the Preferred Message Bytes field. |

| Field | Description |
|---|---|
| Preferred Message Bytes | Specify the preferred number of bytes allowed for the serialized messages in a batch. A message larger than this size results in a larger batch, but the batch size will be equal to or less than the number of bytes you specified in the Absolute Message Bytes field.<br>Oracle recommends that you set this value to 1 MB or less.<br>The value that you enter in this field must be smaller than the value you enter in the Absolute Message Bytes field. |
| Snapshot Interval Size | Defines number of MB per which a snapshot is taken. |

4. Click **Update**.

   The updated settings are saved.

## View Ordering Service Settings

You can view the founder's ordering service settings that were imported into a participant's Oracle Blockchain Platform instance.

If the founder changes the ordering service settings the new settings must be ported to the participant as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service. If there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.

1. Go to the participant's console and select the Network tab.
2. Click **Ordering Service Settings** and click **View**.

   The Ordering Settings dialog is displayed.

4

# Understand and Manage Nodes by Type

This topic contains information to help you understand the different node types and where you can get more information about how the nodes are performing in the network.

**Topics:**



## Manage CA Nodes

This topic contains information about certificate authority (CA) nodes, including how to view and edit the CA node configuration, and how to view the health information for the CA node.

### View and Edit the CA Node Configuration

A certificate authority (CA) node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See CA Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the CA node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

### View Health Information for a CA Node

You can check a certificate authority (CA) node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays the node's performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the CA node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node's performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

# Manage the Console Node

This topic contains information about the console node, including how to view and edit the console node configuration, and how to view the health information for the console node.

## View and Edit the Console Node Configuration

The console node's configuration determines how it performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See Console Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the console node and click its **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

## View Health Information for the Console Node

You can check the console node's metrics to see how it's performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the console node.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node's performance metrics.

   Note the following information:

- If the CPU Utilization percentage is too high, then it might be because too many users are trying to access the console at the same time, or that the console is having technical issues.
- If the utilization percentages are consistently high, then contact Oracle Support

# Manage Orderer Nodes

This topic contains information about ordering service nodes (OSNs), including how to view and edit OSNs, how to view the health information for an OSN, and how to add an additional OSN.

## View and Edit the Orderer Node Configuration

An orderer node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See Orderer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the orderer node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

## View Health Information for an Orderer Node

You can check an orderer node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the orderer node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node's performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support. If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.

## Add an Orderer Node

Founder instances are provisioned with 3 OSNs, all of which are active after instance creation. Additional OSNs can be scaled out as described in Scale Your Instance. These OSNs will not be started automatically. You must start them and export the updated network configuration block to the partipant instances as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

Participant instances are created with 3 OSNs, but none of these OSNs are joined to the network or started when the instance is provisioned. You must follow the instructions in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service in order to join them to the network and start the nodes. If you want to scale out the participant OSNs these steps must be repeated.

# Manage Peer Nodes

This topic contains information about peer nodes, including how to view and edit peer nodes, how to get a list of chaincodes installed on a peer, and how to find health information for a peer node.

## View and Edit the Peer Node Configuration

A peer node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See Peer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the peer node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

## List Chaincodes Installed on a Peer Node

You can view a list of the chaincodes and their versions installed on a specific peer node in your network.

If you don't see the chaincode or the chaincode version you were expecting, then you can install a chaincode or upgrade a chaincode to the peer node. You must be an administrator to install or upgrade a chaincode.

1. Go to the console and select the Nodes tab.

2. In the Nodes tab, click the name of the peer node you want to see information for.

   The Node Information page is displayed.

3. Click the **Chaincodes** pane to view a list of chaincodes installed on the selected peer node.

## View Health Information for a Peer Node

You can check a peer node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization, memory utilization, user transactions endorsed, and user transactions committed.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the peer node you want to see health information for.

   The Node Information page is displayed.

3. Click the **Health** pane to view the node's performance metrics.

   Note the following information:

   - If the CPU Utilization and Memory Utilization percentages are too high, then it might be because the peer is overloaded with endorsement requests. Consider adding another peer or changing the endorsement policy.
   - If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.
   - The User Transactions Endorsed and User Transaction Committed metrics are collected and refreshed every ten minutes. The counts you see are cumulative.
   - If the utilization percentages are consistently high, then contact Oracle Support.

# Manage REST Proxy Nodes

This topic contains information to help you understand how the REST proxy is used, add enrollments to the REST proxy, and view and edit the REST proxy nodes.

## How's the REST Proxy Used?

The REST proxy maps an application identity to a blockchain member, which allows users and applications to call the Oracle Blockchain Platform REST APIs.

Instead of using the native Hyperledger Fabric APIs, Oracle Blockchain Platform can use the REST proxy to interact with the Hyperledger Fabric network. When you use the native Hyperledger Fabric APIs, you connect to the peers and orderer directly. However, the REST proxy allows you to query or invoke a Fabric chaincode through the RESTful protocol.

## Add Enrollments to the REST Proxy

You can add Hyperledger Fabric enrollments to the REST proxy. Enrollments allow users to call the REST proxy without an enrollment certificate.

If you want to add a user to an enrollment, they must already exist in IDCS, and be assigned to the REST_USER role.

Use the Blockchain Platform console to add new enrollments and associate IDCS users with these enrollments. The enrollments are managed entirely within Blockchain Platform, not within IDCS.

For information about how users access the REST resources, see REST API for Oracle Blockchain Platform.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, find the REST proxy node you want to add an enrollment to, and click the **Action** menu for this node.
3. Click **View or Manage Enrollments** to see a list of the node's current enrollments.

   A list of the current enrollments is displayed. You can delete existing enrollments as well as adding new ones from this page.
4. Expand **Create New Enrollment**.
5. In the **Enrollment ID** field, enter the name of the enrollment to add.

   The enrollment ID can include only alphanumeric characters, hyphens (-), and underscores (_).
6. Optionally, in the **User ID** field, enter the ID of a user with the REST_USER role to associate with the enrollment. Click **Enroll**.

   After you click **Enroll**:

   - The enrollment is created and displays in the Enrollments table.
   - The new enrollment is copied to each REST Proxy node in the network.
   - If you specified a user ID, that ID is associated with the enrollment, and cannot be removed from the associated REST users list. If the user ID is not a valid REST user, an error is returned.
   - If you specified a user ID, the generated enrollment certificate includes the ID as the `username` attribute.
   - User IDs that contain a colon (:) are not supported for REST API calls that use basic authentication.
7. In the Associated REST Client Users pane you can view and manage any users associated with a current enrollment, including deleting a user from an enrollment.
8. Add another user to the enrollment by expanding **Associate New Users**. Enter the email or ID of a user that is already assigned the REST_USER role. Click **Associate**.

After you've created an enrollment and associated a user with it, when you use REST to run transactions on the blockchain the initiator listed in the details of the block will be listed as the new enrollment rather that the original default user.

## View and Edit the REST Proxy Node Configuration

A REST proxy node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See REST Proxy Node Attributes.

1. Go to the console and select the Nodes tab.

2. In the Nodes tab, go to the Nodes table, locate the REST proxy node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's **Proposal Wait Time (ms)**, **Transaction Wait Time (ms)**, **Log Level**, and **Transaction Event Logging** attributes as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.

## View Health Information for a REST Proxy Node

You can check a REST proxy node's metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the REST proxy node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node's performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

5

# Extend the Network

This topic contains information to help founders add organizations to the blockchain network. This topic also contains information to help organizations join a network.

**Topics**



## Add Oracle Blockchain Platform Participant Organizations to the Network

This topic contains information about joining an Oracle Blockchain Platform participant organization to an Oracle Blockchain Platform network.

### Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network

Here are the tasks the founder and participants organizations need to perform to set up a blockchain network.

**Adding Participant Organizations to a Blockchain Network**

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Export the participant organization's certificates and import them into the network | Participant organization outputs certificates<br>Founder organization uploads certificates | In the participant organization's instance, use the wizard to output the certificates into a JSON file and send them to the founder organization.<br>The founder uploads the certificates to add the participant to the network. | Import Certificates to Add Organizations to the Network |
| Export the participant organization's ordering service node (OSN) settings and send to the founder administrator | Participant organization outputs a settings file<br>Founder organization uploads the settings | In the participant organization's instance, export the settings into a JSON file and sends them to the founder organization.<br>The founder uploads the settings to add the ordering service. | Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service |

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Export the founder organization's network configuration block and upload it to the participant organization | Founder organization exports network configuration block information<br>Participant organization uploads network configuration block information | In the founder's instance, download the network configuration block information (JSON file).<br>Then in the participant's instance, upload the network configuration block. | Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service |

### Join Participant Organizations to the Channel and Set Anchor Peers

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Create a channel | Founder organization | In the founder's instance, create a channel that the founder and participants use to communicate. Add the founder's peers to the channel.<br>You must select any newly added participants and assign them permissions on the channel.<br>Note that instead of creating a new channel, you can add participants to an existing channel. | Create a Channel |
| Join participants to the channel | Participant organization | In the participant's instance, join the channel that was created in the founder's instance. | Join a Peer to a Channel |
| Set anchor peers on the founder and participants | Founder organization<br>Participant organization | In the founder and participant instances, specify which peers you want to use as anchor peers. You must select at least one anchor peer for each member. | Add an Anchor Peer |

### Deploy the Chaincode Across the Blockchain Network

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Install the chaincode on the founder | Founder organization | In the founder's instance, upload and install the chaincode. Choose the peers to install the chaincode on. | • (Hyperledger Fabric v2.x) Use Quick Deployment<br>• (Hyperledger Fabric v1.4.7)Use Quick Deployment |

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Deploy the chaincode and specify an endorsement policy on the founder | Founder organization | In the founder's instance, deploy the chaincode to activate it on the network.<br>An endorsement policy is required to specify the number of members that must approve chaincode transactions before they're submitted to the ledger. | • (Hyperledger Fabric v2.x) Deploy a Chaincode<br>• (Hyperledger Fabric v1.4.7) Instantiate a Chaincode<br>• (Hyperledger Fabric v2.x) Specify an Endorsement Policy<br>• (Hyperledger Fabric v1.4.7) Specify an Endorsement Policy |
| Install the chaincode on the participant | Participant organization | In the participant's instance, install the chaincode that your network will use.<br>Because you'll install the same chaincode that you installed and deployed on the founder, you don't need to deploy the chaincode on the participant. When the participant installs the chaincode, it's already deployed. | • (Hyperledger Fabric v2.x) Use Quick Deployment<br>• (Hyperledger Fabric v1.4.7)Use Quick Deployment |

**Run Transactions**

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Invoke the chaincode and monitor network activity and ledger updates | Founder organization<br>Participant organization | Begin using your network's chaincode for transactions.<br>Both the founder and the participants can use their consoles to find out information about the activity on the network. Specifically, you can use the console's Channels tab to locate information about specific ledger transactions | • Find Information About Nodes<br>• View a Channel's Ledger Activity |

# Add Fabric Organizations to the Network

This topic contains information about joining Hyperledger Fabric organizations to an Oracle Blockchain Platform network.

# Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network

Here are the tasks that a Fabric organization and the Oracle Blockchain Platform founder organization need to perform to join a Fabric organization to the Oracle Blockchain Platform network.

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Create the certificate file for the Fabric organization | Fabric organization | Find the Fabric organization's Admin, CA, and TLS certificate information and use it to compose a JSON certificates file. | Create a Fabric Organization's Certificates File |
| Upload Fabric organization's certificate file to the Oracle Blockchain Platform network | Founder organization | Use the console to upload and import the Fabric organization's certificate file to add the Fabric organization to the network. | Import Certificates to Add Organizations to the Network |
| Create a channel | Founder organization | Create a new channel and add the Fabric organization to it. | Create a Channel |
| Export the ordering service settings from founder | Founder organization | Output the founder's ordering services settings to a JSON file and send the file to the Fabric organization. | Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service |
| Compose orderer certificate file | Fabric organization | Create a file named orderer.pem that includes the tlscacert information. Go to the exported ordering service settings file and copy the tlscacert information. After you paste the tlscacert information into the orderer.pem file, you must replace all instances of \n with the newline character.<br>The orderer.pem file must have the following format:<br>`-----BEGIN CERTIFICATE-----`<br>`...`<br>`...`<br>`...`<br>`-----END CERTIFICATE-----` | Create a Fabric Organization's Certificates File |

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Provide ordering service settings | Founder organization | Open the ordering service settings file and find the ordering service's address and port and give them to the Fabric organization. For example:<br><br>`"orderingServiceNodes": [ { "address": "grpcs://example_address:7777", ... }]` | NA |
| Add the Fabric organization to the network | Fabric organization | The Fabric organization copies certificates into its environment, sets environment variables, fetches the genesis block, joins the channel, and installs the chaincode. | Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network |

# Create a Fabric Organization's Certificates File

For a Fabric organization to join an Oracle Blockchain Platform network, it must write a certificates file containing its admincerts, cacerts, and tlscacerts information. The Oracle Blockchain Platform founder organization imports this file to add the Fabric organization to the network.

The Fabric certificates information is stored in PEM files located in the Fabric organization's MSP folder. For example, `network_name_example/crypto-config/peerOrganizations/example_org.com/msp/`.

The certificates file must be in written in JSON and must contain the following fields. For all certificates, when you copy the certificate information into the JSON file, you must replace each new line with `\n`, so that the information is all on one line with no spaces, as shown in the following example.

- **mspid** — Specifies the name of the Fabric organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization's Admin certificates file: `Admin@example_org.com-cert.pem`.
- **cacert** — Contains the contents of the organization's CA certificates file: `ca.example_org-cert.pem`.
- **tlscacert** — Contains the contents of the organization's TLS certificate file: `tlsca.example_org-cert.pem`.

- **intermediatecerts**— This optional element contains the contents of an intermediate CA certificates file. Do not specify this element unless there is an intermediate CA certificates file.
- **nodeouidentifiercert**— This section contains certificates that identify Node OU roles.
- **adminouidentifiercert**— Contains the contents of the organization's certificate file that is used to identify Node OU admin roles. If you do not need the admin role, you can use the cacert file contents, or intermediate certificate file contents, as the adminouidentifier contents.
- **clientouidentifiercert**— Contains the contents of the organization's certificate file that is used to identify Node OU client roles.
- **ordererouidentifiercert**— Contains the contents of the organization's certificate file that is used to identify Node OU orderer roles. If you do not need the orderer role, you can use the cacert file contents, or intermediate certificate file contents, as the ordererouidentifier contents.
- **peerouidentifiercert**— Contains the contents of the organization's certificate file used to identify Node OU peer roles.

Structure the file similar to the following example:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "tlscacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
   "nodeouidentifiercert": {
      "adminouidentifiercert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
      "clientouidentifiercert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
      "ordererouidentifiercert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
      "peerouidentifiercert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
    }
  }
}
```

# Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

You must modify the Fabric organization's environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For more information, see Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network.

- The Fabric organization's certificate file was created and sent to the Oracle Blockchain Platform network founder.
- The network founder uploaded the certificates file to add the Fabric organization to the network.
- The network founder created a new channel and added the Fabric organization to it.
- The network founder downloaded its ordering service settings and sent them to the Fabric organization.
- The Fabric organization created the orderer certificate file.
- The network founder gave the ordering service address and port to the Fabric organization.

You must add the Fabric organization and install and test the chaincode.

1. Navigate to the Fabric network directory and launch the peer container.
2. Fetch the channel's genesis block with this command:

   ```
   peer channel fetch 0 mychannel.block -o ${orderer_addr}:${orderer_port} -
   c mychannel --tls --cafile orderer.pem --logging-level debug
   ```

   Where:

   - `{orderer_addr}` is the Founder's orderer address.
   - `{orderer_port}` is the Founder's port number.
   - `-c mychannel` is the name of the channel that the Founder created. This is the channel where the Fabric organization will send and receive transactions on the Oracle Blockchain Platform network.
   - `orderer.pem` is the Founder's orderer certificate file.
3. Join the channel with this command:

   ```
   peer channel join -b mychannel.block -o ${orderer_addr}:${orderer_port} --
   tls --cafile orderer.pem --logging-level debug
   ```

4. Install the chaincode with this command:

   ```
   peer chaincode install -n mycc -v 1.0 -l "golang" -p ${CC_SRC_PATH}
   ```

   Where `CC_SRC_PATH` is the folder that contains the chaincode.

5. Instantiate the chaincode with this command:

```
peer chaincode instantiate -o  ${orderer_addr}:${orderer_port} --
tls --cafile orderer.pem -C mychannel -n mycc -l golang -v 1.0 -c
'{"Args":["init","a","100","b","200"]}' -P <policy_string> --
logging-level debug
```

6. Invoke the chaincode with this command:

```
peer chaincode invoke -o ${orderer_addr}:${orderer_port}  --tls
true --cafile orderer.pem -C mychannel -n mycc -c '{"Args":
["invoke","a","b","10"]}' --logging-level debug
```

7. Query the chaincode with this command:

```
peer chaincode query -C mychannel -n mycc -c '{"Args":
["query","a"]}'  --logging-level debug
```

# Add Organizations with Third-Party Certificates to the Network

This topic contains information about joining organizations using third-party certificates to an Oracle Blockchain Platform network.

## Typical Workflow to Join an Organization With Third-Party Certificates to an Oracle Blockchain Platform Network

Organization with certificates issued by a third-party certificate authority (CA) can join the Oracle Blockchain Platform network as participants.

**Client-only Organizations**

These participants are client-only organizations and have no peers or orderers. They cannot create channels, join peers or install chaincode.

After joining the network, these organizations can use an SDK or a Hyperledger Fabric CLI to:

- Deploy, invoke, and query chaincode if they're a client organization administrator.
- Invoke and query chaincode if they're a client organization non-administrator.

To control who can deploy and invoke chaincode when client-only organizations are part of the network:

- The chaincode owner who installs the chaincode onto peers can decide who can deploy the chaincode by using the Hyperledger Fabric `peer chaincode package -i` instantiation policy command to set the instantiation policy for the chaincode.
- The chaincode instantiator can use the Hyperledger Fabric `peer chaincode instantiate -P` endorsement policy command to set the endorsement policy controlling who can invoke the chaincode.

- The channel owner can decide who can invoke or query a chaincode by setting the channel proposal and query access control list. See Hyperledger Fabric Access Control Lists.

### Workflow

Here are the tasks that an organization with third-party certificates and the Oracle Blockchain Platform founder need to perform to join the organization to an Oracle Blockchain Platform network.

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Get the third-party certificates | Third-party certificates (participant) organization | Go to the third-party CA server and generate the required certificates files. Format the files as needed for import into the network. | Third-Party Certificate Requirements |
| Create the certificates file for import | Third-party certificates (participant) organization | Find the participant's Admin and CA certificate information and use it to compose a JSON certificates file. | Create an Organization's Third-Party Certificates File |
| Upload a certificate file for the third-party (participant) organization | Founder organization | Use the console to upload and import the participant's certificate file to add the participant to the network. | Import Certificates to Add Organizations to the Network |
| Export the ordering service settings from network founder and provide them to the third-party (participant) organization | Founder organization | Output the founder's ordering services settings to a JSON file and send the file to the participant. Open the ordering service settings file and find the ordering service's address and port and give them to the participant. For example:<br><br>`"orderingServiceNodes": [ { "address": "grpcs://example_address:7777" ... }]` | Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service |
| Create the channel | Founder | Create a new channel and add the participant to it. | Create a Channel |
| Install and deploy the chaincode | Founder | In the founder's instance, upload, install, and deploy the chaincode. Choose the network peers to install the chaincode on. | • (Hyperledger Fabric v2.x) Use Quick Deployment<br>• (Hyperledger Fabric v1.4.7)Use Quick Deployment |

| Task | Who Does This? | Description | More Information |
|---|---|---|---|
| Set up the third-party (participant) organization's environment | Third-party certificates (participant) organization | To query or invoke chaincodes, the participant must:<br>• Add the founder's ordering service's address and port to the participant's environment.<br>• Configure the environment to use Hyperledger Fabric CLI or SDKs.<br>• Install the chaincode on peers. | Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network |

# Third-Party Certificate Requirements

To successfully join the network, an organization must generate the required third-party certificates. The information in these certificates is used to create the organization's certificates file, which is then imported into the founder's instance.

**Which Certificates Do Organizations Need to Provide?**

You must generate the following certificates from your CA server:

- Client Public Certificate
- CA Root Certificate

**What Are the Requirements for These Certificates?**

The certificates must meet the following requirements:

- When generating the private key, you must use the Elliptic Curve Digital Signature Algorithm (ECDSA). This algorithm is the only accepted algorithm for Fabric MSP keys.
- The Subject Key Identifier (SKI) is mandatory and you must indicate it as x509 extensions in the extension file.
- You must convert the key files from the .key to the .pem format.
- You must convert the certificates from the .crt to the .pem format.

**Creating the Certificates**

The following walkthrough is an example of how to use OpenSSL or the Hyperledger Fabric cryptogen utility to generate your certificates. For detailed information on the commands used, refer to:

- OpenSSL documentation
- cryptogen utility documentation

To create your certificates using OpenSSL:

1. Create a self-signed CA certificate/key:

```
openssl ecparam -name prime256v1 -genkey -out ca.key
openssl pkcs8 -topk8 -inform PEM -in ca.key -outform pem -nocrypt -out ca-
key.pem
openssl req -new -key ca-key.pem -out ca.csr
openssl x509 -req -days 365 -in ca.csr -signkey ca-key.pem -out ca.crt -
extensions x509_ext -extfile opensslca.conf
openssl x509 -in ca.crt -out ca.pem -outform PEM
```

Our example `opensslca.conf` file:

```
[ req ]
default_bits        = 2048
distinguished_name  = subject
req_extensions      = req_ext
x509_extensions     = x509_ext
string_mask         = utf8only


[ subject ]
countryName         = CN
#countryName_default     = US


stateOrProvinceName     = Beijing
#stateOrProvinceName_default = NY

localityName            = Beijing
#localityName_default        = New York

organizationName         = thirdpartyca, LLC
#organizationName_default    = Example, LLC

# Use a friendly name here because its presented to the user. The
server's DNS
#   names are placed in Subject Alternate Names. Plus, DNS names here is
deprecated
#   by both IETF and CA/Browser Forums. If you place a DNS name here,
then you
#   must include the DNS name in the SAN too (otherwise, Chrome and
others that
#   strictly follow the CA/Browser Baseline Requirements will fail).
commonName          = thirdpartyca
#commonName_default      = Example Company

emailAddress            = ca@thirdpartyca.com



# Section x509_ext is used when generating a self-signed certificate.
I.e., openssl req -x509 ...
[ x509_ext ]
```

```
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer

# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:TRUE
keyUsage            = Certificate Sign, CRL Sign, digitalSignature,
keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage  = serverAuth, clientAuth

# Section req_ext is used when generating a certificate signing
request. I.e., openssl req ...
[ req_ext ]

subjectKeyIdentifier        = hash

basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage  = serverAuth, clientAuth

[ alternate_names ]

DNS.1       = localhost
DNS.2       = thirdpartyca.com
#DNS.3       = mail.example.com
#DNS.4       = ftp.example.com

# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1
```

2. Create a user certificate/key using above CA key:

```
openssl ecparam -name prime256v1 -genkey -out user.key
openssl pkcs8 -topk8 -inform PEM -in user.key -outform pem -nocrypt
-out user-key.pem
openssl req -new -key user-key.pem -out user.csr
```

```
openssl x509 -req -days 365 -sha256 -CA ca.pem -CAkey ca-key.pem -
CAserial ca.srl -CAcreateserial -in user.csr -out user.crt -extensions
x509_ext -extfile openssl.conf
openssl x509 -in user.crt -out user.pem -outform PEM
```

Our example `openssl.conf` file:

```
[ req ]
default_bits        = 2048
default_keyfile     = tls-key.pem
distinguished_name  = subject
req_extensions      = req_ext
x509_extensions     = x509_ext
string_mask         = utf8only

# The Subject DN can be formed using X501 or RFC 4514 (see RFC 4519 for a
description).
# Its sort of a mashup. For example, RFC 4514 does not provide
emailAddress.
[ subject ]
countryName         = CN
#countryName_default     = US

stateOrProvinceName     = Beijing
#stateOrProvinceName_default = NY

localityName            = Beijing
#localityName_default        = New York

organizationName         = thirdpartyca, LLC
#organizationName_default    = Example, LLC

# Use a friendly name here because its presented to the user. The
server's DNS
#   names are placed in Subject Alternate Names. Plus, DNS names here is
deprecated
#   by both IETF and CA/Browser Forums. If you place a DNS name here,
then you
#   must include the DNS name in the SAN too (otherwise, Chrome and
others that
#   strictly follow the CA/Browser Baseline Requirements will fail).
commonName          = admin@thirdpartyca.com
#commonName_default      = Example Company

emailAddress            = admin@thirdpartyca.com
#emailAddress_default        = test@example.com

# Section x509_ext is used when generating a self-signed certificate.
I.e., openssl req -x509 ...
[ x509_ext ]
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer

# You only need digitalSignature below. *If* you don't allow
```

```
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment

subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
#extendedKeyUsage  = Any Extended Key Usage
#extendedKeyUsage = serverAuth, clientAuth


# Section req_ext is used when generating a certificate signing
request. I.e., openssl req ...
[ x509_ca_ext ]
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer


# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:TRUE
keyUsage            = Certificate Sign, CRL Sign, digitalSignature,
keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"


# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
#extendedKeyUsage  = Any Extended Key Usage
extendedKeyUsage = serverAuth, clientAuth


# Section req_ext is used when generating a certificate signing
request. I.e., openssl req ...
[ req_ext ]
subjectKeyIdentifier        = hash
basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"


# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
```

```
#extendedKeyUsage  = Any Extended Key Usage
#extendedKeyUsage = serverAuth, clientAuth

[ alternate_names ]
DNS.1       = localhost
DNS.3       = 127.0.0.1
DNS.4       = 0.0.0.0
# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1
# IPv6 localhost
# DNS.8     = ::1
```

To create your certificates using the Hyperledger Fabric cryptogen utility:

- The following cryptogen commands are used to create Hyperledger Fabric key material:

  ```
  cryptogen generate --config=./crypto-config.yaml
  ```

  Our example `crypto-config.yaml` file:

  ```
  # Copyright IBM Corp. All Rights Reserved.
  #
  # SPDX-License-Identifier: Apache-2.0
  #

  #
  ---------------------------------------------------------------------------
  -
  # "PeerOrgs" - Definition of organizations managing peer nodes
  #
  ---------------------------------------------------------------------------
  -
  PeerOrgs:
    #
  ---------------------------------------------------------------------------
  -
    # Org1
    #
  ---------------------------------------------------------------------------
  -
    - Name: Org1
      Domain: org1.example.com
      EnableNodeOUs: true
      #
  ---------------------------------------------------------------------------
  -
      # "Specs"
      #
  ---------------------------------------------------------------------------
  -
      # Uncomment this section to enable the explicit definition of hosts
  ```

```
in your
    # configuration.  Most users will want to use Template, below
    #
    # Specs is an array of Spec entries.  Each Spec entry consists
of two fields:
    #   - Hostname:   (Required) The desired hostname, sans the
domain.
    #   - CommonName: (Optional) Specifies the template or explicit
override for
    #                 the CN.  By default, this is the template:
    #
    #                              "{{.Hostname}}.{{.Domain}}"
    #
    #                 which obtains its values from the
Spec.Hostname and
    #                 Org.Domain, respectively.
    #
-------------------------------------------------------------------
-------
    # Specs:
    #   - Hostname: foo # implicitly "foo.org1.example.com"
    #     CommonName: foo27.org5.example.com # overrides Hostname-
based FQDN set above
    #   - Hostname: bar
    #   - Hostname: baz
    #
-------------------------------------------------------------------
-------
    # "Template"
    #
-------------------------------------------------------------------
-------
    # Allows for the definition of 1 or more hosts that are created
sequentially
    # from a template. By default, this looks like "peer%d" from 0
to Count-1.
    # You may override the number of nodes (Count), the starting
index (Start)
    # or the template used to construct the name (Hostname).
    #
    # Note: Template and Specs are not mutually exclusive.  You may
define both
    # sections and the aggregate nodes will be created for you.
Take care with
    # name collisions
    #
-------------------------------------------------------------------
-------
    Template:
      Count: 2
      # Start: 5
      # Hostname: {{.Prefix}}{{.Index}} # default
    #
-------------------------------------------------------------------
-------
```

```
    # "Users"
    #
---------------------------------------------------------------------------
-
    # Count: The number of user accounts _in addition_ to Admin
    #
---------------------------------------------------------------------------
-
    Users:
      Count: 1
  #
---------------------------------------------------------------------------
-
  # Org2: See "Org1" for full specification
  #
---------------------------------------------------------------------------
-
  - Name: Org2
    Domain: org2.example.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Users:
      Count: 1
```

**What's Next?**

After confirming that you've outputted and updated the proper files, you can then create the certificates file for import into the Oracle Blockchain Platform network. See Create an Organization's Third-Party Certificates File.

# Create an Organization's Third-Party Certificates File

To join an Oracle Blockchain Platform network, the organization must write a certificates file containing its admincert and cacert information. The network founder imports this file to add the organization to the network.

Go to the certificates files that you generated from the CA server to find the information that you need to create the certificates file. See Third-Party Certificate Requirements.

The certificates file must be in written in JSON and contain the following fields:

- **mspid** — Specifies the name of the organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization's Admin certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **cacert** — Contains the contents of the organization's CA certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.

This is how the file needs to be structured:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
 }
}
```

# Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

You must set up the third-party organization's environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For information, see Typical Workflow to Join an Organization With Third-Party Certificates to an Oracle Blockchain Platform Network.

- The third-party organization’s certificate file was created and sent to the Oracle Blockchain Platform network founder.
- The network founder uploaded the certificates file to add the third-party organization to the network.
- The network founder exported the orderer service's settings and gave the service's address and port to the third-party organization and the organization added them to the environment.
- The network founder created a new channel and added the third-party organization to it.
- The network founder installed and instantiated the chaincode.

**Setup organization's Environment**

Before the third-party organization can successfully use the Oracle Blockchain Platform network, it must set up its environment to use Hyperledger Fabric CLI or SDKs. See the Hyperledger Fabric documentation.

**Install the Chaincode**

The third-party organization must install the chaincode on the peers. These peers must then be joined to the channel so that the chaincode can be invoked.

**Deploy the Chaincode**

If needed, the third-party organizations can deploy the chaincode on the channel. For example:

```
export  CORE_PEER_TLS_ENABLED=true
export  CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export  CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/Admin@customerorg1.com/msp
export  CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input###
CHANNEL_NAME=$1

echo "######### going to instantiate chaincode on channel ${CHANNEL_NAME}
##########"
CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode instantiate
-o ${peer_host}:${port}  --tls $CORE_PEER_TLS_ENABLED --cafile
./tls-ca.pem -C ${CHANNEL_NAME}  -n obcs-example02 -v v0 -c '{"Args":
["init","a","100","b","200"]}'
```

### Invoke the Chaincode

Third-party organizations use the Hyperledger Fabric CLI or SDKs to invoke the chaincode. For example:

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/User1@customerorg1.com/msp
export CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input ###
CHANNEL_NAME=$1

#### do query or invoke on chaincode ####

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode query -C
${CHANNEL_NAME} -n $2 -c '{"Args":["query","a"]}'

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode invoke -o
${peer_host}:${port} --tls $CORE_PEER_TLS_ENABLED --cafile ./tls-
ca.pem -C ${CHANNEL_NAME} -n $2 -c '{"Args":["invoke","a","b","10"]}'
```

6

# Develop Chaincodes

This topic contains information to help you understand how to write and test chaincodes for use in Oracle Blockchain Platform.

**Topics**



## Write a Chaincode

A chaincode is written in Go, Node.js, or Java and then packaged into a ZIP file that is installed on the Oracle Blockchain Platform network.

Chaincodes define the data schema in the ledger, initialize it, perform updates when triggered by applications, and respond to queries. Chaincodes can also post events that allow applications to be notified and perform downstream operations. For example, after purchase orders, invoices, and delivery records have been matched by a chaincode, it can post an event so that a subscribing application can process related payments and update an internal ERP system.

**Resources for Chaincode Development**

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write valid chaincodes.

- Welcome to Hyperledger Fabric. The Key Concepts and Tutorials sections should be read before you write you own chaincode.
- Go Programming Language. The Go compilers, tools, and libraries provide a variety of resources that simplify writing chaincodes.
- Package shim. Package shim provides APIs for the chaincode to access its state variables, transaction context and call other chaincodes. This documents the actual syntax required for your chaincode.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See What Are Chaincode Samples?

You can add rich-query syntax to your chaincodes to query the state database. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

**Package and Zip a Go Chaincode**

Once you've written your chaincode, place it in a ZIP file. You don't need to create a package for the Go chaincode or sign it — the Oracle Blockchain Platform installation and deployment process does this for you as described in Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v2.x) or Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v1.4.7).

If your chaincode has any external dependencies, you can place them in the vendor directory of your ZIP file.

**Vendor the Shim for Go Chaincodes (Hyperledger Fabric v2.x)**

The Go chaincode shim dependency, which was previously included with earlier versions of Hyperledger Fabric, is not included with Hyperledger Fabric v2.x. The shim must now be vendored (imported) to Go chaincodes before they are installed on a peer running Hyperledger Fabric v2.x.

You can use Go modules or a third-party tool such as govendor to vendor the chaincode shim and update it to the version that works with Hyperledger Fabric v2.x.

For more information, see Chaincode shim changes (Go chaincode only) and Upgrade Chaincodes with vendored shim in the Hyperledger Fabric documentation. For more information about Go modules, see Go Modules Reference.

**Package and Zip a Node.js Chaincode**

If you're writing a Node.js chaincode, you need to create a `package.json` file with two sections:

- The `scripts` section declares how to launch the chaincode.
- The `dependencies` section specifies the dependencies.

The following is a sample `package.json` for a Node.js chaincode:

```
{
    "name": "chaincode_example02",
    "version": "1.0.0",
    "description": "chaincode_example02 chaincode implemented in
Node.js",
    "engines": {
        "node": ">=8.4.0",
        "npm": ">=5.3.0"
    },
    "scripts": { "start" : "node chaincode_example02.js" },
    "engine-strict": true,
    "license": "Apache-2.0",
    "dependencies": {
        "fabric-shim": "~1.3.0"
    }
}
```

The packaging rules for a Node.js chaincode are:

- `package.json` must be in the root directory.
- The entry JavaScript file can be located anywhere in the package.
- If `"start" : "node <start>.js"` isn't specified in the `package.json`, `server.js` must be in the root directory.

Place the chaincode and package file in a ZIP file to install it on Oracle Blockchain Platform.

**Package and Zip a Java Chaincode**

If you're writing a Java chaincode, you can choose Gradle or Maven to build the chaincode.

If you're using Gradle, place the chaincode, build.gradle, and settings.gradle in a ZIP file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode package:

```
Archive:  example_gradle.zip
 Length      Date    Time    Name
---------  ---------- -----   ----
      610  02-14-2019 01:36   build.gradle
       54  02-14-2019 01:28   settings.gradle
        0  02-14-2019 01:28   src/
        0  02-14-2019 01:28   src/main/
        0  02-14-2019 01:28   src/main/java/
        0  02-14-2019 01:28   src/main/java/org/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
     5357  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
SimpleChaincode.java
---------                     -------
     6021                     10 files
```

If you're using Maven, place the chaincode and pom.xml in a ZIP file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode package:

```
Archive:  example_maven.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     3313  02-14-2019 01:52   pom.xml
        0  02-14-2019 01:28   src/
        0  02-14-2019 01:28   src/chaincode/
        0  02-14-2019 01:28   src/chaincode/example/
     4281  02-14-2019 01:28   src/chaincode/example/SimpleChaincode.java
---------                     -------
     7594                     5 files
```

### Testing a Chaincode

After you write your chaincode, then you need to test it. See:

- Use a Mock Shim to Test a Chaincode
- Deploy a Chaincode on a Peer to Test the Chaincode

### Installing and Deploying a Chaincode

After you've tested your chaincode, you can deploy it by following the information in Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v2.x) or Typical Workflow to Deploy Chaincodes (Hyperledger Fabric v1.4.7).

### Upgrading a Chaincode

You can upgrade a deployed chaincode by following the steps in Upgrade a Chaincode (Hyperledger Fabric v2.x) or Upgrade a Chaincode (Hyperledger Fabric v1.4.7).

# Use a Mock Shim to Test a Chaincode

This method of testing involves using a mock version of the stub `shim.ChaincodeStubInterface`. With this you can simulate some functionality of your

chaincode before deploying it to Oracle Blockchain Platform. You can also use this library to build unit tests for your chaincode.

1. Create a test file that matches the name of the chaincode file.

   For example, if `car_dealer.go` is the actual implementation code for you smart contract, you would create a test suite called `car_dealer_test.go` containing all the tests for `car_dealer.go`. The test suite filename should be in the `*_test.go` format.

2. Create your package and import statements.

```
package main

import (
    "fmt"
    "testing"

    "github.com/hyperledger/fabric/core/chaincode/shim"
)
```

3. Create your unit test.

```
/*
* TestInvokeInitVehiclePart simulates an initVehiclePart
transaction on the CarDemo cahincode
 */
func TestInvokeInitVehiclePart(t *testing.T) {
    fmt.Println("Entering TestInvokeInitVehiclePart")

    // Instantiate mockStub using CarDemo as the target chaincode
to unit test
    stub := shim.NewMockStub("mockStub", new(CarDemo))
    if stub == nil {
        t.Fatalf("MockStub creation failed")
    }

    var serialNumber = "ser1234"

    // Here we perform a "mock invoke" to invoke the function
"initVehiclePart" method with associated parameters
    // The first parameter is the function we are invoking
    result := stub.MockInvoke("001",
        [][]byte{[]byte("initVehiclePart"),
            []byte(serialNumber),
            []byte("tata"),
            []byte("1502688979"),
            []byte("airbag 2020"),
            []byte("aaimler ag / mercedes")})

    // We expect a shim.ok if all goes well
    if result.Status != shim.OK {
        t.Fatalf("Expected unauthorized user error to be returned")
    }

    // here we validate we can retrieve the vehiclePart object we
```

```
just committed by serianNumber
    valAsbytes, err := stub.GetState(serialNumber)
    if err != nil {
        t.Errorf("Failed to get state for " + serialNumber)
    } else if valAsbytes == nil {
        t.Errorf("Vehicle part does not exist: " + serialNumber)
    }
}
```

> **Note:**
>
> Not all interfaces of the stub are implemented. Stub functions
>
> - `GetQueryResult`
> - `GetHistoryForKey`
>
> are not supported, and attempting to call either of these will result in an error.

# Deploy a Chaincode on a Peer to Test the Chaincode

After you create a chaincode, you must install, deploy, and invoke it to test that it works correctly.

To learn more about writing a chaincode, see Write a Chaincode.

Follow these steps to deploy and test your chaincode.

1. Identify the channel or create a new channel and add peers to it. See Join a Peer to a Channel.
2. Install the chaincode on the peers and deploy it on the channel.
   - (Hyperledger Fabric v2.x) See Use Quick Deployment.
   - (Hyperledger Fabric v1.4.7) See Use Quick Deployment.
3. Use the `Invoke` and `query` REST APIs to test the chaincode with cURL through the REST proxy. See REST API for Oracle Blockchain Platform for descriptions of each endpoint and correct cURL syntax to invoke each operation.
4. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
5. In the channel's **Ledger** pane, view the chaincode's ledger summary.

7

# Build Chaincodes with Low-Code Blockchain App Builder

Blockchain App Builder for Oracle Blockchain Platform is a tool set that assists with rapid development, testing, debugging, and deployment of chaincode on Oracle Blockchain Platform networks, comprising cloud BaaS nodes on Oracle Cloud Infrastructure or on-premises nodes using Enterprise Edition.

A smart contract (also known as a chaincode) defines the different states of a business object between two or more parties and business logic that validates and implements changes as the object moves between these different states. At the heart of every blockchain application is one or more chaincodes. A chaincode must be bug-free and tested before it is deployed.

You can use Blockchain App Builder to generate complex chaincodes in TypeScript (for Node.js chaincode) and Go (for Golang chaincode) from a simple specification file. With the specification file you can specify multiple asset definitions and behaviors. You can then generate and test your chaincodes either on your local system by using a preconfigured instance of Hyperledger Fabric inside Blockchain App Builder, or by connecting to your Oracle Blockchain Platform network.

> **Note:**
>
> Although JavaScript isn't supported by Blockchain App Builder, because TypeScript projects are compiled to JavaScript, you can add basic JavaScript to a TypeScript project if needed.

Blockchain App Builder supports the full development life cycle either from a command-line interface or as an extension for Visual Studio Code.

To get the Blockchain App Builder tools and samples, in the console open the **Developer Tools** tab and select the **Blockchain App Builder** pane. From here you can download the command-line interface tools or the Visual Studio Code extension. Additionally, there are samples - Fabcar, Marbles, Fiat Money Token, Loyalty Token, NFT Art Collection Marketplace, and Fractional NFT in Real Estate - which can be used to see how the tools work or as a template for your own chaincode projects.

**Topics:**



## Using the Blockchain App Builder Command Line Interface

The Blockchain App Builder command line interface helps you build and scaffold a fully-functional chaincode project from a specification file.

After the project is built, you can run and test it on a local Hyperledger Fabric network, or your provisioned Oracle Blockchain Platform network. You can then run SQL rich queries, debug the chaincode, or write and run unit tests using the generated code.

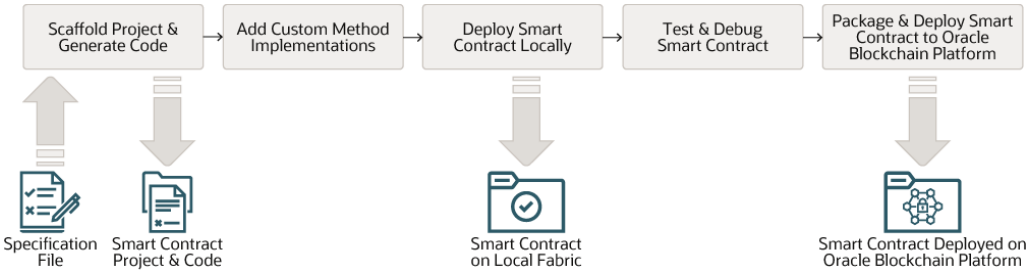



**Table 7-1 Workflow When Using the CLI**

| Task | Description | Related Topics |
|---|---|---|
| Install and configure | Download the Blockchain App Builder CLI from your Oracle Blockchain Platform console and install it and any prerequisite software. | • Install and Configure Blockchain App Builder CLI |
| Create the chaincode project | Create a specification file for the chaincode project. | • Create a Chaincode Project with the Blockchain App Builder CLI |
| Generate the chaincode | Edit the specification file to define the assets and chaincodes to generate, and then run the CLI initialization process to generate your chaincode from the specification file. | Detailed reference information about the structure and contents of the specification file and the generated chaincode project:<br>• Input Specification File<br>• Scaffolded TypeScript Chaincode Project<br>• Scaffolded Go Chaincode Project<br>Detailed information about tokenization support:<br>• Tokenization Support Using Blockchain App Builder<br>• Scaffolded TypeScript Token Project for ERC-1155<br>• Scaffolded Go Token Project for ERC-1155<br>• Scaffolded TypeScript NFT Project for ERC-721<br>• Scaffolded Go NFT Project for ERC-721<br>• Scaffolded TypeScript Project for Token Taxonomy Framework<br>• Scaffolded Go Project for Token Taxonomy Framework |

**Table 7-1 (Cont.) Workflow When Using the CLI**

| Task | Description | Related Topics |
|---|---|---|
| Deploy the chaincode | After your chaincode project is created, you can deploy it locally to the included pre-configured Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition.<br>You can also package the chaincode project for manual deployment to Oracle Blockchain Platform. | • Deploy Your Chaincode to a Local Hyperledger Fabric Network<br>• Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network<br>• Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform |
| Test the chaincode | After your chaincode is running on a network, you can test any of the generated methods.<br>Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network. | • Test Your Chaincode on a Local Hyperledger Fabric Network<br>• Test Your Chaincode on a Remote Oracle Blockchain Platform Network<br>• Execute Berkeley DB SQL Rich Queries |
| Debug the chaincode | The Blockchain App Builder extension for Visual Studio Code includes line-by-line debugging of your chaincode. | • Debugging from Visual Studio Code |
| Synchronize your updates | When you update your specification file, you can synchronize the changes with the generated chaincode files. | • Synchronize Specification File Changes With Generated Source Code |
| Apply patches to the Blockchain App Builder CLI | You can use the `patch` command to apply a patch to the Blockchain App Builder CLI. | • Apply a Patch to the Blockchain App Builder CLI |
| Run unit tests | A basic unit test case setup is included in the project. Additional tests can be added and run. | • Writing Unit Test Cases and Coverage Reports for the Chaincode Project |

# Install and Configure Blockchain App Builder CLI

The following platforms are supported:

- macOS
- Oracle Linux 8.0 or 9.0
- Microsoft Windows 10 or 11

After you've completed the installation process:

- Verify your installation.
- If you're using Go chaincode projects, complete the additional configuration steps.

### Prerequisites

Before you install Blockchain App Builder CLI on your local system, you must install the prerequisites.

> **Note:**
>
> Blockchain App Builder coordinates with Oracle Blockchain Platform and its compilers. If you use any versions of the prerequisites other than the ones mentioned in the following section, deploying your chaincode to a remote Oracle Blockchain Platform network might fail.

When you install Blockchain App Builder, a prerequisites check runs first. If the prerequisites check fails, the installation process is stopped.



## macOS

**Prerequisites**

- Rancher Desktop (tested with 1.4.1). Blockchain App Builder can also work with Docker, but it has been tested and verified with Rancher Desktop. If you plan to use Rancher Desktop, uninstall Docker completely before installing Rancher Desktop. After you install Rancher Desktop, ensure that the container runtime is set to `dockerd (moby)`. To verify the container runtime in Rancher Desktop 1.4.1, click **Kubernetes Settings**, and then **Container Runtime**.
- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0). Do not use versions of Node.js earlier or later than version 18.
- npm v8.x or v9.x (tested with 9.5.0 and 9.5.1)
- Go v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
- If you want to use the synchronization feature of Blockchain App Builder, install Git and then configure your user name and email as shown in the following commands. Specify your user name and email address in the place of `<your_name>` and `<email>`.

```
git config --global user.name "<your_name>"


git config --global user.email "<email>"
```

### Install Node.js and npm by Using nvm

Using nvm to install Node.js and npm gives you the ability to run more commands without `sudo`.

1. Enter the following command to install nvm:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/
install.sh | bash
```

2. Add the following code snippet to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  #
This loads nvm bash_completion
```

3. Log out and then log back in to your operating system.
4. Enter the following command to verify the nvm installation:

```
nvm version
```

5. Enter the following command to install Node.js and npm:

```
nvm install 18.16.0
```

6. Enter the following command to set Node.js 18.16.0 as the default in nvm:

```
nvm alias default 18.16.0
```

## Linux

**Prerequisites**

- Docker v20.10.0 or later
- Docker Compose v1.23.0 or later
- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0). Do not use versions of Node.js earlier or later than version 18.
- npm v8.x or v9.x (tested with 9.5.0 and 9.5.1)
- Go v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
- If you want to use the synchronization feature of Blockchain App Builder, install Git and then configure your user name and email as shown in the following commands. Specify your user name and email address in the place of `<your_name>` and `<email>`.

```
git config --global user.name "<your_name>"
```

```
git config --global user.email "<email>"
```

**Install Node.js and npm by Using nvm**

Using nvm to install Node.js and npm gives you the ability to run more commands without `sudo`.

1. Enter the following command to install nvm:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh
| bash
```

2. Add the following code snippet to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
# This loads nvm bash_completion
```

3. Log out and then log back in to your operating system.
4. Enter the following command to verify the nvm installation:

```
nvm version
```

5. Enter the following command to install Node.js and npm:

```
nvm install 18.16.0
```

6. Enter the following command to set Node.js 18.16.0 as the default in nvm:

```
nvm alias default 18.16.0
```

**Install Docker**

Ensure that `dnf` is updated and pointing to the current repository based on your kernel.

1. Enter the following command to add Docker to the repository list:

```
sudo dnf config-manager --add-repo=https://download.docker.com/
linux/centos/docker-ce.repo
```

2. Enter the following command to install Docker:

```
dnf install docker-ce -y --allowerasing
```

3. Enter the following command to start Docker as a service:

```
sudo systemctl enable --now docker
```

4. Enter the following commands to ensure that the current user has access to Docker:

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

5. Enter the following command to restart the system:

```
sudo reboot
```

**Install Docker Compose**

1. Enter the following `curl` command to get Docker Compose:

```
sudo curl -L https://github.com/docker/compose/releases/download/v2.5.0/
docker-compose-linux-x86_64 -o /usr/local/bin/docker-compos
```

2. Enter the following command to add executable permissions to Docker Compose:

```
sudo chmod +x /usr/local/bin/docker-compose
```

## Windows

**Prerequisites**

- Rancher Desktop (tested with 1.4.1). Blockchain App Builder can also work with Docker, but it has been tested and verified with Rancher Desktop.
- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0). Do not use versions of Node.js earlier or later than version 18.
- npm v8.x or v9.x (tested with 9.5.0 and 9.5.1)
- Go v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
- If you want to use the synchronization feature of Blockchain App Builder, install Git and configure your user name and email as shown in the following commands. Specify your user name and email address in the place of `<your_name>` and `<email>`.

```
git config --global user.name "<your_name>"


git config --global user.email "<email>"
```

**Install Rancher Desktop**

Complete the following steps to install Rancher Desktop on Microsoft Windows.

1. If Docker is installed on your local computer, uninstall it completely.
2. Download and install Rancher Desktop.
3. After the installation wizard completes, before you open Rancher Desktop, run the following commands:

```
wsl --install
wsl --set-default-version 2
wsl --setdefault rancher-desktop
```

4. Open Rancher Desktop to complete the setup process.
5. After you install Rancher Desktop, ensure that the container runtime is set to `dockerd (moby)`. To verify the container runtime in Rancher Desktop 1.4.1, click **Kubernetes Settings**, and then **Container Runtime**.

### Install Blockchain App Builder

Download the Blockchain App Builder CLI package (`oracle-ochain-cli-x.x.x.tgz`) from the **Developer Tools** tab on the **Blockchain App Builder** pane of the Oracle Blockchain Platform console.

- macOS
- Linux
- Windows

## macOS

1. Enter the following command to install Xcode or the XCode command line tools (xcode-select).

   ```
   sudo xcode-select –install
   ```

2. Enter the following command to install Blockchain App Builder (adjust the name of the `.tgz` file for the version that you are installing):

   ```
   npm install -g oracle-ochain-cli-x.x.x.tgz
   ```

Note that Mac OS Catalina can have issues with xcode-select. If you encounter these issues, use the following command to reset and restart xcode-select:

```
xcode-select --reset
```

## Linux

1. Enter the following command to install Blockchain App Builder (adjust the name of the `.tgz` file for the version that you are installing):

   ```
   npm install -g oracle-ochain-cli-x.x.x.tgz
   ```

2. Log out as the current user and then log in again so that group membership takes effect.

## Windows

After you've installed all the prerequisite software, enter the following command to install Blockchain App Builder (adjust the name of the `.tgz` file for the version that you are installing):

```
npm install -g oracle-ochain-cli-x.x.x.tgz
```

### Verify the Installation

In your terminal, type `ochain -v`. The output shows the Blockchain App Builder CLI usage, options, and commands.

### Additional Setup for Go Chaincode Projects

To develop a Go project, you must set the `GOPATH` environment variable. This allows Go to locate your workspace and run your code.

---



## macOS

Before setting the `GOPATH` environment variable, make sure that a `go/` folder exists in your `$HOME` directory. If not, enter the following command to create a `go/` directory in your home directory:

```
mkdir $HOME/go
```

Set your `GOPATH` environment variable by adding the following variables to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

After editing the file, run the following command to make your changes take effect immediately:

```
source ~/.bash_profile
```

Alternately, you can apply the change system-wide by adding the previous variables to the `/etc/bashrc` file.

## Linux

Before setting the `GOPATH` environment variable, make sure that a `go/` folder exists in your `$HOME` directory. If not, enter the following command to create a `go/` directory in your home directory:

```
mkdir $HOME/go
```

Set your `GOPATH` environment variable by adding the following variables to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

After editing the file, run the following command to make your changes take effect immediately:

```
source ~/.bash_profile
```

Alternately, you can apply the change system-wide by adding the previous variables to the `/etc/bashrc` file.

### Windows

Create a `go/` directory in your home directory: `C:\Users\<username>\go`.

## Upgrade Blockchain App Builder CLI

To upgrade Blockchain App Builder, you must uninstall the previous version and then install the newer version.

1. Run the following command to uninstall Blockchain App Builder.

   ```
   npm uninstall -g @oracle/ochain-cli
   ```

2. Verify that Blockchain App Builder is no longer installed by running the following command.

   ```
   ochain -v
   ```

   If Blockchain App Builder is no longer installed, an error message similar to the following text is displayed.

   ```
   -bash: ochain: command not found
   ```

3. Download the latest version of the Blockchain App Builder CLI package (`oracle-ochain-cli-x.x.x.tgz`) from the **Developer Tools** tab on the Blockchain App Builder pane of the Oracle Blockchain Platform console, and then install Blockchain App Builder. For more information, see Install and Configure Blockchain App Builder CLI.

# Create a Chaincode Project with the Blockchain App Builder CLI

To create a chaincode project when using the Blockchain App Builder CLI, you scaffold a chaincode project from a detailed specification file. The generated project then contains all the files than you need.

### Background

Blockchain App Builder's `init` command initializes and scaffolds a ready-to-use chaincode project. Based on simple input, the `init` command can generate complex chaincode projects that include the following features:

- Multiple assets (models) and their behaviors (controllers)
- Auto-generate CRUD (Create/Read/Update/Delete) and non-CRUD methods
- Automatic validation of arguments
- Marshalling/unmarshalling of arguments
- Transparent persistence capability (ORM)
- The ability to call rich queries

The generated project follows the model/controller and decorator pattern, which allows an asset's properties that are maintained on the ledger to be specified as typed fields and extended with specific behaviors and validation rules. This pattern reduces the number of lines of code, which helps in readability and scalability.

### Prerequisites

Before you scaffold a project, you must create an input specification file. For more information, see Input Specification File.

### Scaffolding the Chaincode Project with the `init` Command

Typing `ochain init -h` will list the command usage with all its options. The `init` command has the following options:

- **--cc/-c:**
  The name of the chaincode project. The default value is `MyChaincode`.
- **--lang/-l**
  The language of the scaffolded chaincode. Blockchain App Builder supports Typescript (`ts`) and Go (`go`). If no option is specified, the language defaults to `ts`.
- **--conf/-f** or **--spec**
  The path to an input specification file. Blockchain App Builder reads the input specification file and generates the scaffolded project with many helper tools, which help in reducing the overall development effort. Taking full advantage of the input specification file can significantly reduce the development time.

  The specification file can be in `yaml` or `json` format. If the path is not specified, it defaults to the current directory. See Input Specification File.
- **--out/-o**
  The output directory of the scaffolded chaincode project. If not specified, the scaffolded project is generated in the current directory.

  The output is a fully contained, lightweight, and scalable Typescript or Go chaincode project.

- **--root/-r**
  Valid and required only for Go. The root directory in the *GOHOME* variable for your Go chaincodes. The default value is `example.com`.

**Example**

```
my-mac:~ name$ ochain init --cc MyNewTsProject --lang ts --conf
spec.yml
```

**Defaults**

If no options are specified in the `ochain init` command, the name of the scaffolded project is `MyChaincode` and the language is TypeScript.

The `MyChaincode.model.ts` file contains only one asset, called `MyAsset`, with one property named `value`. The `MyChaincode.controller.ts` file contains one controller with the corresponding CRUD methods for the `MyAsset` model.

**Output**

When the process is complete, you'll have a fully-functional chaincode project that you can deploy either locally or to a remote Oracle Blockchain Platform instance. For a detailed overview of the files created, see:

- Scaffolded TypeScript Chaincode Project
- Scaffolded Go Chaincode Project

For a detailed overview of a token-based project, see also:

- Scaffolded TypeScript Token Project for ERC-1155
- Scaffolded Go Token Project for ERC-1155
- Scaffolded TypeScript NFT Project for ERC-721
- Scaffolded Go NFT Project for ERC-721
- Scaffolded TypeScript Project for Token Taxonomy Framework
- Scaffolded Go Project for Token Taxonomy Framework

## Input Specification File

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

With the specification file you can specify multiple asset definitions and behavior, CRUD and non-CRUD method declaration, custom methods, validation of arguments, auto marshalling/unmarshalling, transparent persistence capability, and invoking rich data queries using SQL SELECTs or CouchDB Query Language. These features will be generated for you.

For information on specifying token assets see the following topics:

- Input Specification File for Token Taxonomy Framework
- Input Specification File for ERC-721
- Input Specification File for ERC-1155

The specification file can be written in either `yaml` or `json`. You can see sample specification files in both formats in the Blockchain App Builder package download:

- `Fabcar-Typescript.yml`
- `Marbles-Go.yml`

> **Note:**
>
> As per Go conventions, exported names begin with a capital letter. Therefore all the asset properties and methods must have names starting with capital letters in the specification file.

### Structure of the Specification File

Typically, you structure a specification file in the following way:

```
assets:
    name:
    type:
    properties:
        name:
        type:
        id:
        derived:
           strategy:
           algorithm:
           format:
        mandatory:
        default:
        validate:
    methods:
        crud:
        others:
customMethods:
```

Blockchain App Builder supports two special asset types, embedded assets and token assets, in addition to generic assets with no specified type. The special assets are defined as `type: embedded` or `type: token` under the `assets:` section of the specification file.

**Table 7-2 Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`assets:`** | This property takes the definition and behavior of the asset. You can give multiple asset definitions here. | |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`name:`** | The name of the asset.<br>The following names are reserved. Do not use these names for assets.<br>• `account`<br>• `role`<br>• `hold`<br>• `token`<br>• `authorization`<br>• `tokenAdmin`<br>• `Account`<br>• `Role`<br>• `Hold`<br>• `Token`<br>• `Authorization`<br>• `TokenAdmin` | `name: owner # Information about the owner` |
| **`type:`** | **Asset types**<br>The following special asset types are supported:<br>• `embedded`<br>• `token`<br>If you do not specify a `type` parameter in the `assets` section, the asset is of the generic type. | |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`type:`**<br>*type: embedded* | If this property is set to `embedded` the asset is defined as an embedded asset. Embedded assets do not have CRUD methods and have to be part of another asset to store in the ledger.<br>In the example, the property `address` is embedded, and is defined in another asset.<br>Embedded assets do not support circular references. For instance, in the previous example the `address` asset cannot contain a reference to the `employee` asset. | Asset: `employee`<br><br>`name: employee`<br>`properties:`<br>`name: employeeId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br><br>`name: firstName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br><br>`name: lastName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br><br>`name: age`<br>`type: number`<br>`validate: positive(),min(18)`<br><br>**`name: address`**<br>**`type: address`**<br><br>Asset: `address`<br><br>**`name: address`**<br><br>**`type: embedded`**<br><br>`properties:`<br>`name: street`<br>`type: string`<br><br>`name: city`<br>`type: string`<br><br>`name: state`<br>`type: string`<br><br>`name: country`<br>`type: string` |
| **`properties:`** | Describe all the properties of an asset. | |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`name:`** | The name of the property. | `name: ownerId # Unique ID for each owner` |
| **`id:`** | • true<br>This specifies the identifier of this asset. This property is mandatory. | `name: owner # Information about the owner`<br>`properties:`<br>`  name: ownerId # Unique ID for each owner`<br>`  type: string`<br>`  mandatory: true`<br>**`  id: true`**<br>`  name: name # Name of the owner`<br>`  type: string`<br>`  mandatory: true` |
| **`type:`** | **Property types**<br>The following basic property types are supported:<br>• `number`<br>• `float`<br>• `string`<br>• `boolean`<br>• `date`<br>• `array`<br>For Go chaincodes, `number` is mapped to `int` and `float` is mapped to `float64`. Other types are not currently supported, including the following types:<br>• `complex`<br>• `unsigned/ signed int`<br>• `8/16/32/64 bits` | `name: year # Model year`<br>**`  type: number`**<br>`  mandatory: true`<br>`  validate: min(1910),max(2020)`<br>`  name: color # Color - no validation as color names are innumerable`<br>**`  type: string`**<br>`  mandatory: true` |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **derived:** | This property specifies that the id property is derived from other keys. Dependent properties should be `string` datatype and not an embedded asset.<br>This property has two mandatory parameters:<br>• `strategy`: takes values of `concat` or `hash`.<br>• `format`: takes an array of specification strings and values to be used by the strategy.<br>Example 1:<br>• The property `employeeID` is dependent on the `firstName` and `lastName` properties.<br>• This property is a concatenation of the values listed in the `format` array.<br>• `IND%1#%2%tIND` is the 0th index in the array and describes the final format.<br>• `%n` is a position specifier that takes its values from the other indexes in the array.<br>• `%t` indicates the value should be `stub.timestamp` from the channel header.<br>• If you need to use the character `%` in the `format` | Example 1<br>`name: employee`<br>`properties:`<br>`name: employeeId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br>**`derived:`**<br>**`strategy: concat`**<br>**`format:`**<br>**`["IND%1#%2%tIND","firstName","lastName"]`**<br>`name: firstName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: lastName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: age`<br>`type: number`<br>`validate: positive(),min(18)`<br>Example 2<br>`name: account`<br>`properties:`<br>`name: accountId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br>**`derived:`**<br>**`strategy: hash`**<br>**`algorithm: 'sha256'`**<br>**`format:`**<br>**`["IND%1#%2%t","bankName","ifsccode"]`**<br>`name: bankName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: ifsccode` |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| | string, it should be escaped with another `%`.<br>• The final format in this example would be: `INDfirstName#lastName1606885454916IND`<br>Example 2:<br>• When using `hash`, you must also use the `algorithm` parameter. The default is `sha256`; `md5` is also supported.<br>• `IND%1#%2%t` is the 0th index in the array and describes the final format.<br>• `%n` is a position specifier that takes its values from the other indexes in the array.<br>• `%t` indicates the value should be `stub.timestamp` from the channel header.<br>• If you need to use the character `%` in the `format` string, it should be escaped with another `%`. | `type: string`<br>`mandatory: true` |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`mandatory:`** | • true<br>• false<br>The corresponding property is mandatory and cannot be skipped while creating an asset. | `name: phone # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`<br>`type: string`<br>**`mandatory: true`**<br>`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`<br>`name: cars # The list of car VINs owned by this owner`<br>`type: string[]`<br>**`mandatory: false`** |
| **`default:`** | This gives you the default value of this property. | |
| **`validate:`** | The given property is validated against some of the out-of-box validations provided by Blockchain App Builder. You can chain validations if you ensure that the chain is valid.<br>If the `validate` property is not provided, then the validation is done against only the property `type`. | |
| **`validate:`**<br>*type: number* | • positive()<br>• negative()<br>• min()<br>• max()<br>These validations can be chained together separated by commas. | `name: offerApplied`<br>`type: number`<br>**`validate: negative(),min(-4)`**<br><br>`name: year # Model year`<br>`type: number`<br>`mandatory: true`<br>**`validate: min(1910),max(2020)`** |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **validate:**<br>*type: string* | • `min()`<br>• `max()`<br>• `email()`<br>• `url()`<br>• `/regex/` - supports PHP regex<br>For Go chaincodes, regular expressions which contain certain reserved characters or whitespace characters should be properly escaped. | `name: website`<br>`type: string`<br>`mandatory: false`<br>**`validate: url()`**<br><br>`name: phone  # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`<br>`type: string`<br>`mandatory: true`<br>**`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`**<br><br>`name: Color #Color can be red, blue, or green`<br>`type: string`<br>`mandatory: true`<br>**`validate: /^\\s*(red\|blue\|green)\\s*$/`** |
| **validate:**<br>*type: boolean* | • true<br>• false<br>In the example, the validation of property `active` is by the type itself (`boolean`) | `name: active`<br>**`type: boolean`** |
| **validate:**<br>*type: array* | By type itself, in the form of `type: number[]`, this conveys that the array is of type number.<br>You can enter limits to the array in the format `number[1:5]` which means minimum length is 1, maximum is 5. If either one is avoided, only min/max is considered. | `name: items`<br>**`type: number[:5]`** |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`validate:`**<br>*type: date* | • `min()`<br>• `max()`<br>Date should be one of these formats:<br>• `YYYY-MM-DD`<br>• `YYYY-MM-DDTHH:MM:SSZ`, where `T` separates the date from the time, and the `Z` indicates UTC. Timezone offsets can replace the `Z` as in -05:00 for Central Daylight Savings Time. | `name: expiryDate`<br>`type: date`<br>**`validate: max('2020-06-26')`**<br><br>`name: completionDate`<br>`type: date`<br>**`validate: min('2020-06-26T02:30:55Z')`** |
| **`methods:`** | Use this to state which of the CRUD (Create/Read/Update/Delete) or additional methods are to be generated.<br>By default, if nothing is entered, all CRUD and other methods are generated. | `methods:`<br>`crud: [create, getById, update, delete]`<br>`others: [getHistoryById, getByRange]` |
| **`crud:`** | • `create`<br>• `getByID` (read)<br>• `update`<br>• `delete`<br>If this array is left empty, no CRUD methods will be created.<br>If the `crud` parameter is not used at all, all four methods will be created by default.<br>The `crud` parameter is not applicable to `token` and `embedded` assets. | `methods:`<br>**`crud: [create, getById, delete]`**<br>`others: [] # no other methods will be created` |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **others:** | • getHistoryById<br>• getByRange<br>getHistoryById returns the history of the asset in a list.<br>getByRange returns all the assets in a given range. For more information, see getByRange (TypeScript) and GetByRange (Go).<br>If this array is left empty, no other methods will be created.<br>If the others parameter is not used at all, both methods will be created by default.<br>The others parameter is not applicable to token and embedded assets. | methods:<br>crud: [create, delete]<br>**others: [] # no other methods will be created**<br><br>methods:<br>crud: [create, getById, update, delete]<br>**others: [getHistoryById, getByRange]** |

**Table 7-2 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`customMethods:`** | This property creates invokable custom method templates in the main controller file. It takes the method signature and creates the function declaration in the controller file.<br>You can provide language specific function declarations here.<br>We provide a custom method named `executeQuery`. If it's added to the specification file, it details how Berkeley DB SQL and CouchDB rich queries can be executed. This method can be invoked only when you are connected to Oracle Blockchain Platform Cloud or Enterprise Edition. | TypeScript<br>`customMethods:`<br>`- executeQuery`<br>`- "buyCar(vin: string, buyerId: string, sellerId: string, price: number, date: Date)"`<br>`- "addCar(vin: string, dealerId: string, price: number, date: Date)"`<br>Go<br>`customMethods:`<br>`- executeQuery`<br>`- "BuyCar(vin string, buyerId string, sellerId string, price int)"`<br>`- "AddCar(vin string, dealerId string, price int)"` |

## Scaffolded TypeScript Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project. The project contains automatically generated classes and functions, CRUD methods, SDK methods, automatic validation of arguments, marshalling/un-marshalling and transparent persistence capability (ORM).

If the chaincode project uses the TypeScript language, the scaffolded project contains three main files:

- `main.ts`
- `<chaincodeName>.model.ts`
- `<chaincodeName>.controller.ts`

All the necessary libraries are installed and packaged. The `tsconfig.json` file contains the necessary configuration to compile and build the TypeScript project.

The `<chaincodeName>.model.ts` file in the `model` subdirectory contains multiple asset definitions and the `<chaincodeName>.controller.ts` file in the `controller` subdirectory contains the assets behavior and CRUD methods.

The various decorators in `model.ts` and `controller.ts` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

**Reference:**

- Models
- Decorators
- ORM
- SDK Methods
- Controller
- Automatically Generated Methods
- Controller Method Details
- Custom Methods
- Init Method

### Models

Every model class extends the `OchainModel` class, which has an additional read-only property called `assetType`. This property can be used to fetch only assets of this type. Any changes to this property are ignored during the creation and updating of the asset. The property value by default is `<modelName>`.

The `OchainModel` class enforces decorator behaviors on properties of the class.

```
@Id('supplierId')
export class Supplier extends OchainModel<Supplier> {
    public readonly assetType = 'supplier';
    @Mandatory()
    @Validate(yup.string())
    public supplierId: string;
```

### Decorators

#### Class decorators

`@Id(identifier)`

This decorator identifies the property which uniquely defines the underlying asset. This property is used as a key of the record, which represents this asset in the chaincode's state. This decorator is automatically applied when a new TypeScript project is scaffolded. The 'identifier' argument of the decorator takes the value from specification file.

```
@Id('supplierId')
export class Supplier extends OchainModel{
...
}
```

**Property decorators**
Multiple property decorators can be used. The decorators are resolved in top to bottom order.

```
@Mandatory()
```

This marks the following property as mandatory so it cannot be skipped while saving to the ledger. If skipped it throws an error.

```
@Mandatory()
public supplierID: string;
```

```
@Default(param)
```

This property can have a default value. The default value in the argument (`param`) is used when the property is skipped while saving to the ledger.

```
@Default('open for business')
@Validate(yup.string())
public remarks: string;
```

```
@Validate(param)
```

The following property is validated against the schema presented in the parameter. The argument `param` takes a yup schema and many schema methods can be chained together. Many complex validations can be added. Refer to https://www.npmjs.com/package/yup for more details.

```
@Validate(yup.number().min(3))
public productsShipped: number;
```

```
@ReadOnly(param)
```

This property decorator marks the underlying property as having a read-only value. The value in the argument, for example *param*, is used when the property is saved in the ledger. Once the value is set it cannot be edited or removed.

```
@ReadOnly('digicur')
public token_name: string;
```

```
@Embedded(PropertyClass)
```

This property decorator marks the underlying property as an embeddable asset. It takes the embeddable class as a parameter. This class should extend the `EmbeddedModel` class. This is validated by the decorator.

In this example, `Employee` has a property called `address` of type `Address`, which is to be embedded with the `Employee` asset. This is denoted by the `@Embedded()` decorator.

```
export class Employee extends OchainModel<Employee> {

   public readonly assetType = 'employee';

   @Mandatory()
   @Validate(yup.string())
   public emplyeeID: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public firstName: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public lastName: string;

   @Validate(yup.number().positive().min(18))
   public age: number;

   @Embedded(Address)
   public address: Address;
}


export class Address extends EmbeddedModel<Address> {

   @Validate(yup.string())
   public street: string;

   @Validate(yup.string())
   public city: string;

   @Validate(yup.string())
   public state: string;

   @Validate(yup.string())
   public country: string;
}
```

When a new instance of the `Address` class is created, all the properties of the `Address` class are automatically validated by the `@Validate()` decorator. Note that the `Address` class does not have the `assetType` property or `@Id()` class decorator. This asset and its properties are not saved in the ledger separately but are saved along with the `Employee` asset. Embedded assets are user defined classes that function as value types. The instance of this class can only be stored in the ledger as a part of the containing object (`OchainModel` assets). All the above decorators are applied automatically based on the input file while scaffolding the project.

```
@Derived(STRATEGY, ALGORITHM, FORMAT)
```

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `STRATEGY`: takes values of `CONCAT` or `HASH`. Requires an additional parameter `ALGORITHM` if `HASH` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `FORMAT`: takes an array of specification strings and values to be used by the strategy.

```
@Id('supplierID')
export class Supplier extends OchainModel<Supplier> {

   public readonly assetType = 'supplier';

   @Mandatory()
   @Derived(STRATEGY.HASH.'sha256',['IND%1IND%2','license','name'])
   @Validate(yup.string())
   public supplierID: string;

   @Validate(yup.string().min(2).max(4))
   public license: string;

   @Validate(yup.string().min(2).max(4))
   public name: string;
```

**Method decorators**

`@Validator(…params)`

This decorator is applied on methods of the main controller class. This decorator is important for parsing the arguments, validating against all the property decorators and returning a model/type object. Controller methods must have this decorator to be invokable. It takes multiple user-created models or yup schemas as parameters.

The order of the parameters must be exactly the same as the order of the arguments in the method.

In the following example, the `Supplier` model reference is passed in the parameter that corresponds to the `asset` type in the method argument. At run time, the decorator parses and converts the method argument to a JSON object, validates against the `Supplier` validators, and after successful validation converts the JSON object to a `Supplier` object and assigns it to the `asset` variable. Then the underlying method is finally called.

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}
```

In the following example, multiple asset references are passed; they correspond to the object types of the method arguments. Notice the order of the parameters.

```
@Validator(Supplier, Manufacturer)
public async createProducts(supplier: Supplier, manufacturer: Manufacturer) {
}
```

Apart from asset references, yup schema objects can also be passed if the arguments are of basic-types. In the following example, `supplierId` and `rawMaterialSupply` are of type `string` and `number` respectively, so the yup schema of similar type and correct order is passed to the decorator. Notice the chaining of yup schema methods.

```
@Validator(yup.string(), yup.number().positive())
public async fetchRawMaterial(supplierID: string, rawMaterialSupply:
number) {
    const supplier = await this.Ctx.Model.get(supplierID, Supplier);
    supplier.rawMaterialAvailable = supplier.rawMaterialAvailable +
rawMaterialSupply;
    return await this.Ctx.Model.update(supplier);
}
```

### ORM

Transparent Persistence Capability or simplified ORM is captured in the `Model` class of the Context (`Ctx`) object. If your model calls any of the following SDK methods, access them by using `this.Ctx.Model`.

SDK methods that implement ORM are the following methods:

- `save` – this calls the Hyperledger Fabric `putState` method
- `get` – this calls the Hyperledger Fabric `getState` method
- `update` – this calls the Hyperledger Fabric `putState` method
- `delete` – this calls the Hyperledger Fabric `deleteState` method
- `history` – this calls the Hyperledger Fabric `getHistoryForKey` method
- `getByRange` – this calls the Hyperledger Fabric `getStateByRange` method
- `getByRangeWithPagination` – this calls the Hyperledger Fabric `getStateByRangeWithPagination` method

For more information, see: SDK Methods.

### SDK Methods

> **Note:**
>
> Beginning with version 21.3.2, the way to access the ORM methods has changed. Run the `ochain --version` command to determine the version of Blockchain App Builder.

In previous releases, the ORM methods were inherited from the `OchainModel` class. In version 21.3.2 and later, the methods are defined on the `Model` class of Context (`Ctx`) object. To call these methods, access them by using `this.Ctx.Model.<method_name>`.

The following example shows a method call in previous releases:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier){
```

```
    return await asset.save();
}
```

The following example shows a method call from the version 21.3.2 and later:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
      return await this.Ctx.Model.save(asset);
}
```

After you upgrade to version 21.3.2, make this change in all chaincode projects that you created with an earlier version of Blockchain App Builder. If you use the `sync` command to synchronize changes between the specification file and your source code, the changes are automatically brought to your controller for the ready-to-use methods. You still need to manually resolve any conflicts.

**save**

The `save` method adds the caller `asset` details to the ledger.

This method calls the Hyperledger Fabric `putState` internally. All marshalling/unmarshalling is handled internally. The `save` method is part of the `Model` class, which you access by using the `Ctx` object.

```
Ctx.Model.save(asset: <Instance of Asset Class> , extraMetadata?: any) :
Promise <any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}
```

**get**

The `get` method is a method of `OchainModel` class which is inherited by the concrete model classes of `{chaincodeName}.model.ts`. The `get` method is part of the `Model` class, which you access by using the `Ctx` object.

If you would like to return any asset by the given `id`, use the generic controller method `getAssetById`.

```
Ctx.Model.get(id: string, modelName: <Model Asset Class Name>) :
Promise<asset>
```

Parameters:

- `id : string` – Key used to save data into the ledger.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise: <Asset>` - If the `modelName` parameter is not provided and data exists in ledger, then `Promise<object>` is returned. If the `id` parameter does not exist in ledger, an error message is returned. If the `modelName` parameter is provided, then an object of type `<Asset>` is returned. Even though any asset with given `id` is returned from the ledger, this method will take care of casting into the caller `Asset` type. If the asset returned from the ledger is not of the `Asset` type, then it throws an error. This check is done by the read-only `assetType` property in the `Model` class.

Example:

```
@Validator(yup.string())
public async getSupplierById(id: string) {
    const asset = await this.Ctx.Model.get(id, Supplier);
    return asset;
}
```

In the example, `asset` is of the type `Supplier`.

**`update`**
The `update` method updates the caller `asset` details in the ledger. This method returns a promise.

This method calls the Hyperledger Fabric `putState` internally. All the marshalling/ unmarshalling is handled internally. The `update` method is part of the `Model` class, which you can access by using the `Ctx` object.

```
Ctx.Model.update(asset: <Instance of Asset Class> , extraMetadata?:
any) : Promise <any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await this.Ctx.Model.update(asset);
}
```

**delete**

This deletes the asset from the ledger given by `id` if it exists. This method calls the Hyperledger Fabric `deleteState` method internally. The `delete` method is part of the `Model` class, which you can access by using the `Ctx` object.

```
Ctx.Model.delete(id: string): Promise <any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any>` - Returns a promise on completion.

Example:

```
@Validator(yup.string())
public async deleteSupplier(id: string) {
    const result = await this.Ctx.Model.delete(id);
    return result;
}
```

**history**

The `history` method is part of the `Model` class, which you can access by using the `Ctx` object. This method returns the asset history given by `id` from the ledger, if it exists.

This method calls the Hyperledger Fabric `getHistoryForKey` method internally.

```
Ctx.Model.history(id: string): Promise <any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any[]>` - Returns any [] on completion.

Example

```
@Validator(yup.string())
public async getSupplierHistoryById(id: string) {
    const result = await this.Ctx.Model.history(id);
```

```
    return result;
}
```

Example of the returned asset history for `getSupplierHistoryById`:

```
[
    {
        "trxId":
"8ef4eae6389e9d592a475c47d7d9fe6253618ca3ae0bcf77b5de57be6d6c3829",
        "timeStamp": 1602568005,
        "isDelete": false,
        "value": {
            "assetType": "supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 10,
            "license": "abcdabcdabcd",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    },
    {
        "trxId":
"92c772ce41ab75aec2c05d17d7ca9238ce85c33795308296eabfd41ad34e1499",
        "timeStamp": 1602568147,
        "isDelete": false,
        "value": {
            "assetType": "supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 15,
            "license": "valid license",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    }
]
```

**getByRange**

The `getByRange` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRange` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ] >`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, use the generic controller method `getAssetsByRange`.

```
Ctx.Model.getByRange(startId: string, endId: string, modelName: <Asset Model
Class Name> ): Promise <any>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

Example:

```
@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string) {
    const result = await this.Ctx.Model.getByRange(startId, endId, Supplier);
    return result;
}
```

**getByRangeWithPagination**

The `getByRangeWithPagination` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRangeWithPagination` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ] >`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, filtered by page size and bookmarks, use the generic controller method `getAssetsByRange`.

```
public async getByRangeWithPagination<T extends OchainModel<T>>(startId:
string, endId: string, pageSize: number, bookmark?: string, instance?: new
(data: any, skipMandatoryCheck: boolean, skipReadOnlyCheck: boolean) => T):
Promise<T[]>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded from the range.

- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

**`getId`**
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` – Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
@Validator(yup.string(), yup.string())

public async customGetterForSupplier(license: string, name: string){
    let object = {
      license : license,
      name: name
    }
    const id = await this.Ctx.Model.getID(object);
    return this.Ctx.Model.get(id);
}
```

For token SDK methods, see the topics under Tokenization Support Using Blockchain App Builder.

### Controller

Main controller class extends `OchainController`. There is only one main controller.

```
export class TSProjectController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable from outside, the rest of them are hidden.

### Automatically Generated Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}

@Validator(yup.string())
public async getSupplierById(id: string) {
    const asset = await this.Ctx.Model.get(id, Supplier);
    return asset;
}

@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await this.Ctx.Model.update(asset);
}

@Validator(yup.string())
public async deleteSupplier(id: string) {
    const result = await this.Ctx.Model.delete(id);
    return result;
}

@Validator(yup.string())
public async getSupplierHistoryById(id: string) {
    const result = await this.Ctx.Model.history(id);
    return result;
}

@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string) {
    const result = await this.Ctx.Model.getByRange(startId, endId, Supplier);
    return result;
}
```

### Controller Method Details

Apart from the above model CRUD and non-CRUD methods, Blockchain App Builder provides out-of-the box support for other Hyperledger Fabric methods from our controller. These methods are:

- `getAssetById`
- `getAssetsByRange`
- `getAssetHistoryById`
- `query`
- `queryWithPagination`
- `generateCompositeKey`

- `getByCompositeKey`
- `getTransactionId`
- `getTransactionTimestamp`
- `getChannelID`
- `getCreator`
- `getSignedProposal`
- `getArgs`
- `getStringArgs`
- `getMspID`
- `getNetworkStub`

> **Note:**
>
> These methods are available with the `this` context in any class that extends the `OChainController` class.

For example:

```
public async getModelById(id: string) {
    const asset = await this.getAssetById(id);
    return asset;
}
@Validator(yup.string(), yup.string())
public async getModelsByRange(startId: string, endId: string) {
    const asset = await this.getAssetsByRange(startId, endId);
    return asset;
}
public async getModelHistoryById(id: string) {
    const result = await this.getAssetHistoryById(id);
    return result;
}
```

**`getAssetById`**

The `getAssetById` method returns asset based on `id` provided. This is a generic method and be used to get asset of any type.

```
this.getAssetById(id: string): Promise<byte[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <byte [ ]>` - Returns promise on completion. You have to convert `byte[]` into an object.

**getAssetsByRange**

The `getAssetsByRange` method returns all assets present from `startId` (inclusive) to `endId` (exclusive) irrespective of asset types. This is a generic method and can be used to get assets of any type.

```
this.getAssetsByRange(startId: string, endId: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< shim.Iterators.StateQueryIterator>` - Returns an iterator on completion. You have to iterate over it.

**getAssetHistoryById**

The `getAssetHistoryById` method returns history iterator of an asset for `id` provided.

```
this.getAssetHistoryById(id: string):
Promise<shim.Iterators.HistoryQueryIterator>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise<shim.Iterators.HistoryQueryIterator>` - Returns a history query iterator. You have to iterate over it.

**query**

The `query` method will run a Rich SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
this.query(queryStr: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `queryStr : string` - Rich SQL/Couch DB query.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

**queryWithPagination**
This method runs a Rich SQL/Couch DB query over the ledger, filtered by page size and bookmarks. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
public async queryWithPagination(query: string, pageSize: number,
bookmark?: string)
```

Parameters:

- `query : string` - Rich SQL/Couch DB query.
- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

**generateCompositeKey**
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
this.generateCompositeKey(indexName: string, attributes:
string[]): string
```

Parameters:

- `indexName : string` - Object Type of the key used to save data into the ledger.
- `attributes: string[ ]` - Attributes based on which composite key will be formed.

Returns:

- `string` - Returns a composite key.

**getByCompositeKey**
This method returns the asset that matches the key and the column given in the attribute parameter while creating composite key. `indexOfId` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`. Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
this.getByCompositeKey(key: string, columns: string[],
indexOfId: number): Promise<any []>
```

Parameters:

- `key: string` – Key used to save data into ledger.

- `columns: string[ ]` - Attributes based on key is generated.
- `indexOfId: number` - Index of attribute to be retrieved from Key.

Returns:

- `Promise< any [ ]` - Returns any `[]` on completion.

**`getTransactionId`**
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
this.getTransactionId(): string
```

Parameters:

- none

Returns:

- `string` - Returns the transaction ID for the current chaincode invocation request.

**`getTransactionTimestamp`**
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
this.getTransactionTimestamp(): Timestamp
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Timestamp` - Returns the timestamp when the transaction was created.

**`getChannelID`**
Returns the channel ID for the proposal for chaincode to process.

```
this.getChannelID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the channel ID.

**`getCreator`**
Returns the identity object of the chaincode invocation's submitter.

```
this.getCreator(): shim.SerializedIdentity
```

Parameters:

- none

Returns:

- `shim.SerializedIdentity` - Returns identity object.

**`getSignedProposal`**
Returns a fully decoded object of the signed transaction proposal.

```
this.getSignedProposal():
shim.ChaincodeProposal.SignedProposal
```

Parameters:

- none

Returns:

- `shim.ChaincodeProposal.SignedProposal` - Returns decoded object of the signed transaction proposal.

**`getArgs`**
Returns the arguments as array of strings from the chaincode invocation request.

```
this.getArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

**`getStringArgs`**
Returns the arguments as array of strings from the chaincode invocation request.

```
this.getStringArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

**`getMspID`**
Returns the MSP ID of the invoking identity.

```
this.getMspID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

**`getNetworkStub`**
The user can get access to the shim stub by calling `getNetworkStub` method. This will help user to write its own implementation of working directly with the assets.

```
this.getNetworkStub(): shim.ChaincodeStub
```

Parameters:

- none

Returns:

- `shim.ChaincodeStub` - Returns chaincode network stub.

**`invokeCrossChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
this.invokeCrossChaincode(chaincodeName: string, methodName: string, args:
string[], channelName: string): Promise<any>
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `Promise<any>` - Returns a JSON object that contains three fields:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

**`invokeChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
this.invokeChaincode(chaincodeName: string, methodName: string, args:
string[], channelName: string): Promise<any>
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `Promise<any>` - Returns a JSON object that contains three fields:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

### Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

```
/**
*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP
CS/EE network.
*
*/
@Validator(yup.string()}
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
@Validator(yup.string(), yup.number()}
public async fetchRawMaterial(supplierId: string, rawMaterialSupply:
number) {
}
@Validator(yup.string(), yup.string(), yup.number())
public async getRawMaterialFromSupplier(manufacturerId: string,
supplierId: string, rawMaterialSupply: number) {
}
@Validator(yup.string(), yup.number(), yup.number())
```

```
public async createProducts(manufacturerId: string, rawMaterialConsumed:
number, productsCreated: number) {
}
public async sendProductsToDistribution() {
}
```

**Init Method**

A custom `init` method is provided in the controller with an empty definition. If you use Blockchain App Builder to deploy or upgrade, the `init` method is called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v1.4.7 platform, the `init` method is also called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v2.x platform, you must call the `init` method manually. You can use a third-party tool such as Postman to call the `init` method manually.

```
export class TestTsProjectController extends OchainController {
    public async init(params: any) {
        return;
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

## Scaffolded Go Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project. The project contains automatically generated classes and functions, CRUD methods, SDK methods, automatic validation of arguments, marshalling/un-marshalling and transparent persistence capability (ORM).

If the chaincode project is in the Go language, the scaffolded project contains three main files:

- `main.go`
- `<chaincodeName>.model.go`
- `<chaincodeName>.controller.go`

All the necessary libraries are installed and packaged.

The `<chaincodeName>.model.go` file in the `model` subdirectory contains multiple asset definitions and the `<chaincodeName>.controller.go` file in the `controller` subdirectory contains the asset's behavior and CRUD methods. The various Go struct tags and packages in `model.go` and `controller.go` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

The scaffolded project can be found in `$GOPATH/src/example.com/<chaincodeName>`

**Reference:**

- Model
- Validators
- ORM

- SDK Methods
- Composite Key Methods
- Stub Method
- Other Methods
- Utility Package
- Controller
- Automatically Generated Methods
- Custom Methods
- Init Method

### Model

#### Asset Type Property

By default every struct will have an additional property called `AssetType`. This property can be useful in fetching only assets of this type. Any changes to this property is ignored during create and update of asset. The property value by default is `<modelName>`.

```
type Supplier struct {
AssetType string 'json:"AssetType" default:"TestGoProject.Supplier"'

SupplierId            string      'json:"SupplierId"
validate:"string,mandatory" id:"true'
RawMaterialAvailable  int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
License               string      'json:"License"
validate:"string,min=10"'
ExpiryDate            date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
Active                bool      'json:"Active" validate:"bool"
default:"true"'
Metadata              interface{} 'json:"Metadata,omitempty"'
}
```

#### Validators

**Id**

`id:"true"`

This validator identifies the property which uniquely defines the underlying asset. The asset is saved by the value in this key. This validator automatically applies when a new Go project is scaffolded.

In the below screenshot `"SupplierId"` is the key for the supplier asset and has a tag property `id:"true"` for the SupplierId property.

```
type Supplier struct {
    SupplierId              string        'json:"SupplierId"
```

```
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int           'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string        'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date     'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool          'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{}   'json:"Metadata,omitempty"'
}
```

### Derived

```
derived:"strategy,algorithm,format"
```

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `strategy`: takes values of `concat` or `hash`. Requires an additional parameter `algorithm` if `hash` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `format`: takes an array of specification strings and values to be used by the strategy.

```
type Supplier struct{
   AssetType string 'json:"AssetType" final:"chaincode1.Supplier"'
   SupplierId   string  'json:"SupplierId" validate:"string" id:"true"
mandatory:"true"
derived:"strategy=hash,algorith=sha256,format=IND%1%2,License,Name"'
   Name         string  'json:"Name" validate:"string,min=2,max=4"'
   License      string  'json:"License" validate:"string,min=2,max=4"'
}
```

### Mandatory

```
validate:"mandatory"
```

This marks the following property as mandatory and cannot be skipped while saving to the ledger. If skipped it throws an error. In the below example, `"SupplierId"` has a `validate:"mandatory"` tag.

```
Type Supplier struct {
    Supplierld              string      'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string      'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool        'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{} 'json:"Metadata,omitempty"'
}
```

**Default**
`default:"<param>"`

This states that the following property can have a default value. The default value in the default tag is used when the property is skipped while saving to the ledger. In the below example property, `Active` has a default value of `true`, provided as tag `default:"true"`

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

**Validate types**
Basic Go types are validated for a property by defining a validate tag. These are the validate tags based on types:

- string: `validate: "string"`
- date: `validate: "date"`
- number: `validate: "int"`
- boolean: `validate: "bool"`

**Min validator**
`validate:"min=<param>"`

Using the min validator, minimum value can be set for a property of type number and string.

For type int: In the example, `RawMaterialAvailable` property has a minimum value of 0 and if a value less than 0 is applied to `RawMaterialAvailable` an error will be returned.

For type string: For the string type minimum validator will check the length of the string with the provided value. Therefore, in the below example the `License` property has to be minimum 10 characters long.

Example:

```
Type Supplier struct {
    Supplierld             string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable   int          'json:"RawMaterialAvailable"
```

```
validate:"int,min=0"'
    License                string       'json:"License"
validate:"string,min=10"'
    ExpiryDate             date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                 bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata               interface{}  'json:"Metadata,omitempty"'
}
```

**Max validator**

`validate:"max=<param>"`

Using the max validator, the maximum value can be set for a property of type number and string.

For type int: Like the min validator, for type int, if a value provided for the `structfield` is greater than the value provided in the validator then an error will be returned.

For type string: Like the min validator, max validator will also check the length of the string with given value. In the example, the `Domain` property has a maximum value of 50, so if the `Domain` property has a string length more than 50 characters, then an error message will be returned.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold" validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=1-5"'
    Domain             string        'json:"Domain"
validate:"url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

**Date validators**
**Before validator:**

`validate:"before=<param>"`

The before validator validates a property of type `date` to have a value less than the specified in parameter.

In this example, the `ExpiryDate` property should be before `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

**After validator:**

```
validate:"after=<param>"
```

The before validator validates a property of type `date` to have a value greater than the specified in parameter.

In this example, the `CompletionDate` property should be after `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Manufacturerld        string       'json:"Manufacturerld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,max=8"'
    ProductsAvailable     int          'json:"ProductsAvailable"
validate:"int"'
    CompletionDate        date.Date    'json:"CompletionDate"
validate:"date,after=2020-06-26"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

**URL validator**

```
validate:"url"
```

The URL validator will validate a property for URL strings.

In this example, the `Domain` property has to be a valid URL.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
```

```
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold" validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=1-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

**Regexp validator**

`validate:"regexp=<param>"`

Regexp validator will validate property for the input regular expression.

In this example, the `PhoneNumber` property will validate for a mobile number as per the regular expression.

```
type Customer struct {
Customerld      string       'json:"Customerld" validate:"string,mandatory"
id:"true"'
Name            string       'json:"Name" validate:"string,mandatory"'
ProductsBought  int          'json:"ProductsBought" validate:"int"'
OfferApplied    int          'json:"OfferApplied" validate :"int,nax=0"'
PhoneNumber     string       'json:"PhoneNumber" validate:"string,regexp=A\(?
([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$"'
Received        bool         'json:"Received" validate:"bool"'
Metadata        interface{}  'json:"Metadata,omitempty"'
}
```

**Multiple validators**

Multiple validators can be applied a property.

In this example, the `Domain` property has validation for a string, URL, and min and max string length.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold" validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=1-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
```

```
    Metadata           interface{}    'json:"Metadata,omitempty"'
}
```

### ORM

Transparent Persistence Capability or simplified ORM is captured in the `Model` class of the Context (`Ctx`) object. If your model calls any of the following SDK methods, access them by using `t.Ctx.Model`.

SDK methods that implement ORM are the following methods:

- `Save` – this calls the Hyperledger Fabric `PutState` method
- `Get` – this calls the Hyperledger Fabric `GetState` method
- `Update` – this calls the Hyperledger Fabric `PutState` method
- `Delete` – this calls the Hyperledger Fabric `DeleteState` method
- `History` – this calls the Hyperledger Fabric `GetHistoryForKey` method
- `GetByRange` – this calls the Hyperledger Fabric `GetStateByRange` method
- `GetByRangeWithPagination` – this calls the Hyperledger Fabric `GetStateByRangeWithPagination` method

### SDK Methods

Go chaincodes implement Transparent Persistence Capability (ORM) with the model package.

> **Note:**
>
> Beginning with version 21.2.3, the way to access the ORM methods has changed. Run the `ochain --version` command to determine the version of Blockchain App Builder.

In previous releases, the ORM methods were exposed as static methods in the model package. The methods are now defined on the model receiver, which holds the transaction stub. To call these methods, you use the model receiver held by the transaction context in the controller. You call these methods as `t.Ctx.Model.<method_name>` instead of `model.<method_name>`.

The following example shows `Save` and `Get` method calls in previous releases:

```
func (t *Controller) CreateSupplier(asset Supplier) (interface{},
error) {
  return model.Save(&asset)
}

func (t *Controller) GetSupplierById(id string) (Supplier, error) {
  var asset Supplier
  _, err := model.Get(id, &asset)
  return asset, err
}
```

The following example shows `Save` and `Get` method calls from the version 21.2.3 and later:

```
func (t *Controller) CreateSupplier(asset Supplier) (interface{}, error) {
  return t.Ctx.Model.Save(&asset)
}

func (t *Controller) GetSupplierById(id string) (Supplier, error) {
  var asset Supplier
  _, err := t.Ctx.Model.Get(id, &asset)
  return asset, err
}
```

After you upgrade to version 21.2.3, make this change in all chaincode projects that you created with an earlier version of Blockchain App Builder. If you use the `sync` command to synchronize changes between the specification file and your source code, the changes are automatically brought to your controller for the ready-to-use methods. You still need to manually resolve any conflicts.

The following ORM methods are exposed via the model package:

**`Get`**
Queries the ledger for the stored asset based on the given ID.

```
func Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `Id` - The ID of the asset which is required from the ledger.
- `result (interface{})` - This is an empty asset object of a particular type, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.
- `asset (interface)` - Empty asset object, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.

Returns:

- `interface {}` - Interface contains the asset in the form of `map[string]interface{}`. Before operating on this map, it is required to assert the obtained interface with type `map[string]interface{}`. To convert this map into an asset object, you can use the utility API `util.ConvertMaptoStruct` (see: Utility Package).
- `error` - Contains an error if returned, or is nil.

**`Update`**
Updates the provided asset in the ledger with the new values.

```
func Update(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj (interface)` - The object that is required to be updated in the ledger is passed by reference into this API with the new values. The input asset is validated and verified according to the struct tags mentioned in the model specification and then stored into the ledger.

Returns:

- `interface{}` - The saved asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

**Save**
Saves the asset to the ledger after validating on all the struct tags.

```
func Save(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj/args[0] (interface{})` - The object that needs to be stored in the ledger is passed by reference in this utility method.
- `metadata/args[1] (interface{})` - This parameter is optional. It has been given in order to facilitate you if you're required to store any metadata into the ledger along with the asset at the runtime. This parameter can be skipped if no such requirement exists.

Returns:

- `interface {}` - The asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

**Delete**
Deletes the asset from the ledger.

```
func Delete(Id string) (interface{}, error)
```

Parameters:

- `id (string)` - The ID of the asset which is required to be deleted from the ledger.

Returns:

- `interface {}` - Contains the asset being deleted in the form of `map[string]interface{}`.

**GetByRange**
Returns the list of assets by range of IDs.

```
func GetByRange(startKey string, endKey string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey (string)` - Starting ID for the range of objects which are required.

- `endkey (string)` - End of the range of objects which are required.
- `asset interface` - (optional) Empty array of assets, which is passed by reference. This array will contain the result from this method. To be used if type-specific result is required.

Returns:

- `[]map[string]interface{}` - This array contains the list of assets obtained from the ledger. You can access the objects iterating over this array and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains an error if returned, or is nil.

**`GetByRangeWithPagination`**

The `GetByRangeWithPagination` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`, filtered by page size and bookmark. This method calls the Hyperledger Fabric `GetStateByRangeWithPagination` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ] >`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, filtered by page size and bookmarks, use the generic controller method `getAssetsByRange`.

```
func (m *Model) GetByRangeWithPagination(startKey string, endKey string,
pageSize int32, bookmark string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey : string` – Starting key of the range. Included in the range.
- `endkey : string` – Ending key of the range. Excluded from the range.
- `pageSize : number` – The page size of the query.
- `Bookmark : string` – The bookmark of the query. Output starts from this bookmark.
- `asset interface` – (Optional) An empty array of assets, passed by reference. This array will contain the result from this method. Use this parameter to get type-specific results.

Returns:

- `[]map[string]interface{}` – An array that contains the list of assets retrieved from the ledger. You can access the objects by iterating over this array and asserting the objects as `map[string]interface{}` and using a utility for conversion to an asset object.
- `error` – Contains an error if an error is returned, otherwise nil.

**GetHistoryById**
Returns the history of the asset with the given ID.

```
func GetHistoryByID(Id string) ([]interface{}, error)
```

Parameters:

- `Id (string)` - ID of the asset for which the history is needed.

Returns:

- `[]interface{}` - This slice contains the history of the asset obtained from the ledger in form of slice of `map[string]interface{}`. You can access each history element by iterating over this slice and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains the error if observed.

**Query**
The query method will run a SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func Query(queryString string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Input the query string.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

**QueryWithPagination**
The query method will run a SQL/Couch DB query over the ledger, filtered by page size and bookmark. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func (m *Model) QueryWithPagination(queryString string, pageSize
int32, bookmark string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Rich SQL/Couch DB query.
- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

**`InvokeCrossChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
func InvokeCrossChaincode(chaincodeName string, method string, args
[]string, channelName string) (interface{}, error)
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `interface{}` - Returns a `map[string]interface{}` object that contains three keys:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

Return Value Example:

```
{
      "isValid": true,
      "message": "Successfully invoked method [CreateAccount] on sub-
chaincode [erc721_go_453]",
      "payload": {
                  "AccountId":
"oaccount~6b83b8ab931f99442897dd04cd7a2a55f808686f49052a40334afe3753fda4c4",
                  "AssetType": "oaccount",
                  "BapAccountVersion": 0,
                  "NoOfNfts": 0,
                  "OrgId": "appdev",
                  "TokenType": "nonfungible",
                  "UserId": "user2"
      }
}
```

**InvokeChaincode**

You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
func InvokeChaincode(chaincodeName string, method string, args
[]string, channelName string) (interface{}, error)
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `interface{}` - Returns a `map[string]interface{}` object that contains three keys:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, in UTF-8 format.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

Return Value Example:

```
{
    "isValid": true,
    "message": "Successfully invoked method [CreateAccount] on sub-
chaincode [erc721_go_453]",
    "payload":
"{\"AssetType\":\"oaccount\",\"AccountId\":\"oaccount~c6bd7f8dcc339bf71
44ea2e1cf953f8c1df2f28482b87ad7895ac29e7613a58f\",\"UserId\":\"user1\",
\"OrgId\":\"appdev\",\"TokenType\":\"nonfungible\",\"NoOfNfts\":0,\"Bap
AccountVersion\":0}"
}
```

### Composite Key Methods

**GenerateCompositeKey**

This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
func GenerateCompositeKey(indexName string, attributes []string)
(string, error)
```

Parameters:

- `indexName (string)` - Object type of the composite key.

- `attrbutes ([]string)` - Attributes of the asset based on which the composite key has to be formed.

Returns:

- `string` - This contains the composite key result.
- `error` - Contains the error if observed.

**`GetByCompositeKey`**
This method returns the asset that matches the key and the column given in the parameters. The `index` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
func GetByCompositeKey(key string, columns []string, index int)
(interface{}, error)
```

Parameters:

- `key (string)` - Object type provided while creating composite key.
- `column ([]string)` - This is the slice of attributes on which the ledger has to be queried using the composite key.
- `index(int)` - Index of the attribute.

Returns:

- `Interface{}` - Contains the list of assets which are result of this method.
- `error` - Contains any errors if present.

### Stub Method

**`GetNetworkStub`**
This method will return the Hyperledger Fabric `chaincodeStub`.

You can get access to the shim stub by calling the `GetNetworkStub` method. This will help you write your own implementation working directly with the assets.

```
func GetNetworkStub() shim.ChaincodeStubInterface
```

Parameters:

- none

Returns:

- `shim.ChaincodeStubInterface` - This is the Hyperledger Fabric chaincode stub.

### Other Methods

- `GetTransactionId()`
- `GetTransactionTimestamp()`

- `GetChannelID()`
- `GetCreator()`
- `GetSignedProposal()`
- `GetArgs()`
- `GetStringArgs()`
- `GetCreatorMspId()`
- `GetId`

**`GetTransactionId`**
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
func GetTransactionId() string
```

Parameters:

- none

Returns:

- `string` - This contains the required transaction ID.

**`GetTransactionTimestamp`**
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
func GetTransactionTimestamp() (*timestamp.Timestamp, error)
```

Parameters:

- none

Returns:

- `timestamp.Timestamp` - Contains the timestamp required.
- `error` - Contains any errors if present.

**`GetChannelID`**
Returns the channel ID for the proposal for the chaincode to process.

```
func GetChannelID() string
```

Parameters:

- none

Returns:

- `string` - Contains the required channel ID as a string.

**GetCreator**
Returns the identity object of the chaincode invocation's submitter

```
func GetCreator() ([]byte, error)
```

Parameters:

- none

Returns:

- `[]byte` - Contains the required identity object serialized.
- `error` - Contains any errors if present.

**GetSignedProposal**
Returns a fully decoded object of the signed transaction proposal.

```
func GetSignedProposal() (*peer.SignedProposal, error)
```

Parameters:

- none

Returns:

- `*peer.SignedProposal` - Contains the required signed proposal object.
- `error` - Contains any errors if present.

**GetArgs**
Returns the arguments as array of strings from the chaincode invocation request.

```
func GetArgs() [][]byte
```

Parameters:

- none

Returns:

- `[][]byte` - Contains the arguments passed.

**GetStringArgs**
Returns the arguments intended for the chaincode Init and Invoke as a string array.

```
func GetStringArgs() []string
```

Parameters:

- none

Returns:

- `[]string` - Contains the required arguments as a string array.

**GetCreatorMspId**
Returns the MSP ID of the invoking identity.

```
func GetCreatorMspId() string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

**GetId**
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` - Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
func (t *Controller) CustomGetterForSupplier(License string, Name
string)(interface{}, error){
   var asset Supplier
   asset.License = License
   asset.Name = Name
   id,err := t.Ctx.Model.GetId(&asset)

   if err !=nil {
      return nil, fmt.Errorf("error in getting ID %v", err.Error())
   }
   return t.GetSupplierById(id)
}
```

### Utility Package

The following methods in the utility package may be useful:

**Util.CreateModel**
Parses the provided JSON string and creates an asset object of the provided type.

```
func CreateModel(obj interface{}, inputString string) error
```

Parameters:

- `inputString (string)` - The input JSON string from which the object is to be created.

- `obj (interface{})` - The reference of the object that is to be created from the JSON string. This object will store the created model which is also validated as per validator tags.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

**`util.ConvertMapToStruct`**
Convert the provided map into object of provided type.

```
func ConvertMapToStruct(inputMap map[string](interface{}), resultStruct
interface{}) error
```

Parameters:

- `inputMap (map[string](interface{}))` - Map which needs to be converted into the asset object.
- `resultStruct (interface{})` - The reference of the required asset object which needs to be generated from the map. Contains the result asset object required.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

For token SDK methods, see the topics under Tokenization Support Using Blockchain App Builder.

### Controller

The `Controller.go` file implements the CRUD and custom methods for the assets.

You can create any number of classes, functions, or files, but only those methods that are defined on chaincode struct are invokable from outside, the rest of them are hidden.

### Automatically Generated Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
//
//Supplier
//
func (t *ChainCode) CreateSupplier(inputString string) (interface{}, error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputString)
    if err != nil {
        return nil, err
    }
    return model.Save(&obj)
}

func (t *ChainCode) GetSupplierById(id string) (interface{}, error) {
    asset, err := model.Get(id)
    return asset, err
```

```
}

func (t *ChainCode) UpdateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputstring)
    if err != nil {
        return nil, err
    }
return model.Update(&obj)
}

func (t *ChainCode) DeleteSupplier(id string) (interface{}, error) {
    return model.Delete(id)
}

func (t *ChainCode) GetSupplierHistoryById(id string) (interface{},
error) {
    historyArray, err := model.GetHistoryByld(id)
    return historyArray, err
}

func (t *ChainCode) GetSupplierByRange(startkey string, endKey string)
(interface{}, error) {
    assetArray, err := model.GetByRange(startkey, endKey)
    return assetArray, err
}
```

### Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

You can implement the functionality according to the business logic. If you add custom methods, add them to the controller file. If you add custom methods to the library instead of the controller file, your changes will be lost when the library folder contents are updated during the synchronization or chaincode upgrade processes.

```
//
//Custom Methods
//
/*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP
CS/EE network.
*/
func (t *ChainCode) ExecuteQuery(inputQuery string) (interface{},
error) {
    resultArray, err := model.Query(inputQuery)
    return resultArray, err
}
```

```
func (t *ChainCode) FetchRawMaterial(supplierId string, rawMaterialSupply
int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) GetRawMaterialFromSupplier(manufacturerId string,
supplierId string, rawMaterialSupply int) (interface{} error) {
    return nil, nil
}

Func (t *ChainCode) CreateProducts(manufacturerId string,
rawMaterialConsumed int, productsCreated int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) SendProductsToDistribution() (interface{}, error) {
    return nil, nil
}
```

For Go chaincodes, every custom method should return two values: *empty interface*, *error*. For example:

```
func (t *Controller) FetchRawMaterial(supplierId string, rawMaterialSupply
int) (interface{}, error) {
    return nil, nil
}
```

#### Init Method

A custom `Init` method is provided in the controller with an empty definition. If you use Blockchain App Builder to deploy or upgrade, the `Init` method is called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v1.4.7 platform, the `Init` method is also called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v2.x platform, you must call the `Init` method manually. You can use a third-party tool such as Postman to call the `Init` method manually.

```
type Controller struct {
}
func (t *Controller) Init(args string) (interface{}, error)
    { return nil, nil
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

## Deploy Your Chaincode Using the CLI

Once your chaincode project is created, you can deploy it locally to the automatically generated Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.

# Deploy Your Chaincode to a Local Hyperledger Fabric Network

Once you have created your chaincode project, you can deploy it to a local Hyperledger Fabric network. This single-channel test network is created for you when you install Blockchain App Builder.

The Blockchain App Builder `ochain run` command starts the Hyperledger Fabric network, other services, and installs and deploys the chaincode for you.

```
my-mac:GOProject myname$ ochain run -h
Usage: run [options] [...args]
Run chaincode project locally in debug mode.

Arguments :
[...args] (optional) Chaincode instantiate arguments. Arguments should
be space separated.

Options:
-h, --help         output command usage information
-D, --debug        enable debug logging
-P, --debug-port   (optional) specify debug process port
-b, --build        (optional) rebuild runtime if already exists
-p, --project      (optional) Path to Chaincode project to run. If not
specified, it defaults to current directory.

Examples :
$> ochain run
```

### Token Projects

For token chaincode projects, you must specify a list of admins with the `ochain run` command:

```
ochain run <adminList array>
```

The `adminList` array is an array of information that specifies the token admins. The `adminList` array is a mandatory parameter when you deploy a token chaincode project for the first time. If you deploy the project again, you can pass an empty array for the `adminList` parameter or you can use the `adminList` parameter to add token admins. Other deployers who are not the first-time deployer must supply an empty array for the `adminList` parameter. The parameter information is different for fungible tokens and non-fungible tokens:

- For fungible tokens that use the Token Taxonomy Framework standard, the parameters are `org_id` and `user_id`.
- For any tokens that use the ERC-1155 standard, the parameters are `orgId` and `userId`.
- For non-fungible tokens that use the ERC-721 standard and a TypeScript project, the parameters are `orgId` and `userId`.
- For non-fungible tokens that use the ERC-721 standard and a Go project, the parameters are `OrgId` and `UserId`.

The following examples are for non-fungible tokens.

Example `adminList` array for TypeScript on Mac OSX and Linux:

```
'[{"userId":"userid", "orgId":"OrgMSPId"}]'
```

Example `adminList` array for Go on Mac OSX and Linux:

```
'[{"UserId":"userid", "OrgId":"OrgMSPId"}]'
```

Example `adminList` array for TypeScript on Microsoft Windows:

```
"[{\"userId\":\"userid\", \"orgId\":\"OrgMSPId\"}]"
```

Example `adminList` array for Go on Microsoft Windows:

```
"[{\"UserId\":\"userid\", \"OrgId\":\"OrgMSPId\"}]"
```

For the local Hyperledger Fabric network, the `OrgMSPId` field is fixed to the value `Org1MSP`.

If you would like to see the debug logs, you can pass the `--debug` option to the command. On Windows, use Command Prompt instead of PowerShell if you specify the `--debug` option. You can run the basic network and deploy the chaincode on a different port from the default by passing the `--port` option to the command.

**Verifying**

The following logs show that the chaincode has been installed and deployed successfully.

```
my-mac:TSProject myname$ ochain run
Recreating orderer.example.com ... done
Recreating ca.example.com ... done
Recreating peer0.org1.example.com ... done
[2020-09-23T18: 04:09.132] [INFO] default -
============ Started Install Chaincode ============
[2020-09-23T18:04:09.193] [INFO] default Chaincode TSProject:1 not installed.
[2020-09-23T18:04:09.317] [INFO] default - Successfully sent install
Proposal and received ProposalResponse
[2020-09-23T18:04:09.317] [INFO] default - Successfully installed chaincode
TSProject
[2020-09-23T18:04:09.317] [INFO] default -
============ Finished Install Chaincode ============
[2020-09-23T18:04:09.317] [INFO] default - Successfully installed chaincode
TSProject
[2020-09-23T18:04:09.318] [INFO] default -
============ started instantiate Chaincode ============
[2020-09-23T18:04:09.366] [INFO] default - Successfully sent Proposal and
received ProposalResponse
[2020-09-23T18:04:11.434] [INFO] default - The chaincode instantiate
transaction has been committed on peer localhost:7051
[2020-09-23T18:04:11.434] [INFO] default - The chaincode instantiate
transaction was valid.
[2020-09-23T18:04:11.435] [INFO] default - Successfully sent transaction to
```

```
the orderer.
[2020-09-23T18:04:11.435] [INFO] default - Successfully instantiated
chaincode TSProject on channel mychannel
[2020-09-23T18:04:11.435] [INFO] default -
============ Finished instantiate Chaincode ============
[2020-09-23T18:04:11.4351 INFO] default - Successfully instantiated
chaincode TSProject on channel mychannel
INFO (Runtime): Chaincode TSProject installed and ready:
INFO (RunCommand): Chaincode TSProject deployed
```

### Troubleshooting

You may encounter the following issues when running your chaincode project on a local network.

**Missing Go permissions**
While installing Go chaincode project in local network, you might see an error similar to the following:

```
My-Mac:GoProj myname$ ochain run
Starting ca.example.com    ... done
Starting orderer.example.com ... done
Starting peer0.orgl.example.com ... done
INFO (Runtime): 2020/06/22 22:57:09 build started

INFO (Runtime): Building ....

INFO (Runtime): go build runtime/cgo: copying /Users/myname/Library/
Caches/go-build/f8/….….d: open /usr/local/go/pkg/darwin_amd64/
runtine/
cgo.a: permission denied

ERROR (Runtime): go build runtine/cgo: copying /Users/myname/Library/
Caches/go-build/f8/….….d: open /usr/local/go/pkg/darwin_amd64/runtime/
cgo.a: permission denied

INFO (Runtime): An error occurred while building: exit status 1

Stopping peer0.orgl.exmple.com ... done
Stopping ca.example.com    ... done
Stopping orderer.example.con ... done
```

This is due to missing permissions for Go. This error has been seen only in Mac OS. This is a known issue:

- https://github.com/golang/go/issues/37962
- https://github.com/golang/go/issues/24674
- https://github.com/udhos/update-golang/issues/15

Solution: change the permissions of your `$GOROOT` and try `ochain run` again:

```
sudo chmod -R 777 /usr/local/go
```

**Deployment failure**
Due to deployment failure, corrupt deployment, Docker peer container full, or Docker peer was killed in local network, you may see an error similar to:

```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating Chaincode
GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container: Failed to
generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error (404):
manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting container:
Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API error
(404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown exited:
signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again.

Solution: try the `ochain run` command again, but with the `-b` option. This option rebuilds the runtime for you.

```
ochain run -b
```

**Environment Rebuild Required**
Rebuild your environment if you see a `channel not found` error or an error similar to the following text:

```
Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com:
error while creating mount source path '/host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/
runtime/network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com... error
```

```
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/crypto-config/peerOrganizations/org1.example.com/ca': mkdir /
host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount
source path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/crypto-config/peerOrganizations/org1.example.com/ca': mkdir /
host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount
source path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com ... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/crypto-config/peerOrganizations/org1.example.com/ca': mkdir /
host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount
source path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/crypto-config/peerOrganizations/org1.example.com/ca': mkdir /
host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Error in Chaincode deployment
```

To rebuild your local environment, run the following command:

```
ochain run -b
```

## Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network

After you've deployed and tested your chaincode project on a local network to ensure it's working as designed, you can deploy it to Oracle Blockchain Platform.

### Deploy Your Chaincode

Usage: `ochain deploy [options] [...args]`

The following arguments and options can be used with the `ochain deploy` command:

```
my-mac:TSProject myname$ ochain deploy -h
Usage: deploy [options] [...args]
Deploy chaincode project to Oracle Blockchain Platform

Arguments:
     [...args]    (optional) Chaincode instantiate arguments. Arguments
should be space separated.

Options :
    -h, --help                      output command usage information
    -D, --debug                     enable debug logging
    -P, --project <project>         (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -c, --channel <channel>         (optional) Blockchain Channel to deploy
chaincode to. If not specified, it defaults to the 'default' channel.
    -u, --username <username>       (required) A user name that has install
chaincode privileges. Contact your administrator for more details.
    -v, --userversion <userversion> (optional) A user-specified chaincode
version.
                            If a version isn't specified, for a new
chaincode it will start at v1 and then increment to v2, v3, and so on.
                            For an existing chaincode, v1.a will increment
to v1.a1, v1 will increment to v2, and v1.0 will increment to v1.1.
    -s --sign <password>            (required) Password to authenticate user.
    -r --url <url>                  (required) URL of the remote OBP
instance. Example: https://<blockchain-instance-url>:7443/

Examples:
$> ochain deploy -u <username> -s <password> -r <url of the remote OBP
instance> -c <name of the channel>
```

Enter the Oracle Identity Cloud Service user name and password for an Oracle Blockchain Platform user with the `ADMIN` or `REST_CLIENT` roles. For more information about users and roles, see Set Up Users and Application Roles.

To invoke the chaincode, only the `REST_CLIENT` role is necessary. To deploy or upgrade the chaincode, the IDCS user must also be assigned the `ADMIN` role.

After the chaincode has successfully deployed to the remote Oracle Blockchain Platform, the log will show that the following events occurred:

- Oracle Blockchain Platform details were successfully fetched.
- The list of peers was successfully fetched.
- The chaincode project was successfully installed.
- The chaincode project was successfully approved and committed.
- The chaincode was successfully deployed on each peer and the channel.

In an environment with multiple organizations, to re-deploy the chaincode on the same channel through a participant instance, use the console to deploy the chaincode.

#### Upgrading the Chaincode Project

Upgrading the chaincode is handled automatically by Blockchain App Builder. After you make changes to your chaincode, call the `ochain deploy` command again, which will automatically upgrade the project for you. When you run the `ochain deploy` command again, specify an empty array for the `adminList` parameter or use the `adminList` parameter to add token admins. If you are not the first-time deployer, you must supply an empty array for the `adminList` parameter.

If your upgrade is successful, the log will show that the following events occurred:

- Oracle Blockchain Platform details were successfully fetched.
- The list of peers was successfully fetched.
- A check was made to determine if the chaincode project is already installed, and if so, the version was fetched.
- The chaincode version was successfully upgraded (for example, from version 1.0 to 2.0).

In an environment with multiple organizations, to upgrade the chaincode, use the console and manually approve the chaincode from the participants.

## Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

You can package your chaincode projects for manual deployment to Oracle Blockchain Platform Cloud or Enterprise Edition.

Usage: `ochain package`

The `package` command creates an archive file that contains only the build and distribution files. The `binary`, `libs`, `node_modules`, and `test` folders from your chaincode project are not included. This archive file can be manually uploaded to Oracle Blockchain Platform for deployment.

Because of changes to software prerequisites, when you run the `ochain package` command for TypeScript chaincode, you are prompted for the provisioning date of the Oracle Blockchain Platform instance that you want to create the package for. The TypeScript chaincode created in Blockchain App Builder is not compatible with previous versions of Oracle Blockchain Platform without changes to the underlying

infrastructure. Blockchain App Builder packages the chaincode infrastructure accordingly based on your selection.

```
my-mac:~ myname$ ochain package -h
Usage: package [options]
Package and archive an Ochain chaincode project
Options :
    -h, --help             output command usage information
    -D, --debug            enable debug logging
    -p, --project <path>   Path to the Ochain chaincode project to be
packaged. If not specified, it defaults to current directory.
    -o, --out <path>       Path to the generated chaincode archive file. If
not specified, it defaults to current directory.
About:
This CLI command packages and archives an existing chaincode project
Examples:
$> ochain package --project <Path to the Ochain chaicode project> —out <Path
to the generated chaincode archive file>
```

When the command completes successfully, the location of the package is returned.

This command takes two optional arguments:

- `--project`
  This option defines the location of the Blockchain App Builder chaincode project to package. If not specified, the location defaults to the current directory.
- `--out`
  This option can be used to give the output path of the generated archive file. If not specified, it defaults to the current directory.

Example:

```
ochain package -p /Blockchain/DevTools/bp1/CC -o /Blockchain/DevTools/bp1/
output

"Your chaincode project has been packaged at /Blockchain/DevTools/bp1/output/
CC.zip"
```

# Test Your Chaincode Using the CLI

If your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

## Test Your Chaincode on a Local Hyperledger Fabric Network

Once your chaincode project is running on a local network, you can test it.

Open a new shell and navigate to the project directory to interact with your chaincodes. After a chaincode is installed and deployed, you can submit transactions to the functions inside your chaincode by using the `ochain invoke` and `ochain query` commands.

**ochain invoke**

Usage: `ochain invoke <methodName> <methodArguments>`

The following are arguments and options taken by the `ochain invoke` command:

```
my-mac:TSProject myname$ ochain invoke -h
Usage: invoke [options] <methodName> [...args]
Invoke a Chaincode transaction

Arguments :
    <methodName>  (required) Name of chaincode method to invoke.
    [...args]     (optional) Chaincode method input parameters if any.
Parameters should be space separated strings/JSON strings for objects.

Options:
    -h, --help                output command usage information
    -D, --debug               enable debug logging
    -P, --project <path>      (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -c, --channel <channel>   (optional) Blockchain Channel to deploy
chaincode to. If not specified, it defaults to the 'default' channel.
    -u, --username <username> (optional, if -r option is applied) A
user name that has install chaincode privileges. Contact your
administrator for more details.
    -s, --sign <password>     (optional) Password to authenticate user.
    -r, --url <url>           (required) URL of the remote OBP
instance. Example: https://<blockchain-instance-url>:7443/

Examples:
$> ochain invoke <method>
(without chaincode initial arguments)
$> ochain invoke <method>
'{"manufacturerId":"m01","rawMaterialAvailable":9,"productsAvailable":4
,"completionDate":"05-26-2020"}'
(for a single parameter)
$> ochain invoke <method> 's01' 's10'
$> ochain invoke <method> 's01'
'{"manufacturerId":"m01","rawMaterialAvailable":9,"productsAvailable":4
,"completionDate":"05-26-2020"}'
(for multiple parameters)
$> ochain invoke <method> 's01' 's10' -r <url of the remote OBP
instance> -u <username> -s <password>
(for remote invocation)
```

### Mac OSX and Linux

If the method takes one argument, enter it as a string. For example:

```
ochain invoke createSupplier
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}'
```

Another example:

```
ochain invoke getSupplierDetails 's01'
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}'
```

If the method takes more than one argument, they should be separated by a space. For example:

```
ochain invoke getSupplierByRange 's01' 's03'
```

If you have embedded assets in your chaincode such as an employee asset which uses an embedded address asset:

```
name: employee
  properties:
     name: employeeId
     type: string
     mandatory: true
     id: true

     name: firstName
     type: string
     validate: max(30)
     mandatory: true

     name: lastName
     type: string
     validate: max(30)
     mandatory: true

     name: age
     type: number
     validate: positive(),min(18)

     name: address
     type: address


name: address

type: embedded

properties:
   name: street
   type: string

   name: city
   type: string

   name: state
   type: string
```

```
  name: country
  type: string
```

You would use something similar to the following to invoke the chaincode:

```
ochain invoke createEmployee '{"employeeID":"e01", "firstName":"John",
"lastName":"Doe",
"age":35, "address":{"street":"Elm Ave", "city":"LA",
"state":"California", "country":"US"}}'
```

### Windows

Windows command prompt doesn't accept single quotes ('), so all arguments have to be kept in double quotes ("). Any argument that contains a double quote must be escaped.

For example:

```
ochain invoke createSupplier
"{\"supplierId\":\"s01\",\"rawMaterialAvailable\":5,\"license\":\"valid
supplier\",\"expiryDate\":\"2020-05-30\",\"active\":true}"
```

If the method takes more than one argument, they should be separated by a space. For example:

```
ochain invoke getSupplierByRange "s01" "s03"
```

If you have embedded assets in your chaincode such as an employee asset which uses an embedded address asset as shown above, you can use something similar to the following to invoke the chaincode:

```
ochain invoke createEmployee "{\"employeeID\":\"e01\",
\"firstName\":\"John\",
\"lastName\":\"Doe\", \"age\":35, \"address\":{\"street\":\"Elm Ave\",
\"city\":\"LA\",
\"state\":\"California\", \"country\":\"US\"}}"
```

### Validations

The method arguments are validated against the validations specified in the specification file. If any validation fails, errors will be listed in the output.

When it invokes successfully it should display a log similar to:

```
========== Started Invoke Chaincode ==========
[2020-06-23T18:37:54.563] [INFO] default - Successfully sent Proposal
and received ProposalResponse
[2020-06-23T18:37:56.619] [INFO default - The chaincode invoke
transaction has been committed on peer localhost:7051
[2020-06-23T18:37:56.619] [INFO] default - The chaincode invoke
transaction was valid.
[2020-06-23T18:37:56.620] [INFO default - Successfully sent
transaction to the orderer.
```

```
[2020-06-23T18:37:56.620] [INFO] default - Successfully invoked method
"createSupplier" on chaincode "TSProject" on channel "mychannel"
[2020-06-23T18:37:56.620] [INFO] default -
========== Finished Invoke Chaincode ==========
```

**ochain query**

Usage: `ochain query <methodName> <methodArguments>`

Following are the arguments and options taken by the `ochain query` command:

```
my-mac:TSProject myname$ ochain query -h
Usage: query [options] <methodName> [...args]
Invoke a Chaincode Query.

Arguments :
    <methodName>     (required) Name of chaincode method to invoke.
    [...args]        (optional) Chaincode method input parameters if any.
Parameters should be space separated strings/JSON strings for objects.

Options:
    -h, --help                output command usage information
    -D, --debug               enable debug logging
    -P, --project <path>      (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -c, --channel <channel>   (optional) Blockchain Channel to deploy
chaincode to. If not specified, it defaults to the 'default' channel.
    -u, --username <username> (optional, if -r option is applied) A user
name that has install chaincode privileges. Contact your administrator for
more details.
    -s, --sign <password>     (optional) Password to authenticate user.
    -r, --url <url>           (required) URL of the remote OBP instance.
Example: https://<blockchain-instance-url>:7443/

Examples:
$> ochain query <method>
(without chaincode initial arguments)
$> ochain query <method> s01
(for a single parameter)
$> ochain query <method> 's01' 's10'
$> ochain query <method> 's01' '{"manufacturerId":"m01"}'
(for multiple parameters)
$> ochain query <method> 's01' 's10' -r <url of the remote OBP instance> -u
<username> -s <password>
(for remote query)
```

The `ochain query` command follows the same rules of passing `<methodName>` and `<methodArguments>` as `ochain invoke`.

- On Mac OSX and Linux, single quotes can be used and there's no need to escape quotes within arguments.
- On Windows, all arguments must surrounded by double quotes and any quote within an argument must be escaped.

### Testing Multiple Token Users Locally

To test a token project with multiple users locally, you can use the `tokenUser` property to change the caller of each transaction. Every scaffolded chaincode project includes a `.ochain.json` file, which stores metadata of the chaincode. You change the caller by updating the value of `tokenUser` field in the `.ochain.json` file.

```
{
  "name": "digiCurrCC",
  "description": "Chaincode package for digiCurrCC",
  "chaincodeName": "digiCurrCC",
  "chaincodeType": "node",
  "configFileLocation": "/Users/user1/token.yml",
  "appBuilderVersion": "21.2.3",
  "nodeVersion": "v12.18.1",
  "tokenUser": "admin"
}
```

When a project is scaffolded, the `tokenUser` property is set to the default `admin` user of the local network. To change the caller of a transaction, change the `tokenUser` property to match the `user_id` property that was set when the account was created when the `createAccount` (TypeScript) or `CreateAccount` (Go) method was called.

### Automatic Installation and Deployment After Update

Whenever you update your chaincode, the changes will be compiled, installed and deployed automatically when it's deployed to a local network. There is no need to strip down or bring up the local network again. All projects will be automatically compiled and deployed on every change.

## Test Your Chaincode on a Remote Oracle Blockchain Platform Network

After your chaincode project has successfully deployed to your remote Oracle Blockchain Platform network, you can test it as described in Test Your Chaincode on a Local Hyperledger Fabric Network.

If you deployed your chaincode manually, instead of using Blockchain App Builder, you must call the `init` function manually before you test your chaincode.

You can use the same `ochain invoke` and `ochain query` commands to perform all method transactions on a remote Oracle Blockchain Platform Cloud or Enterprise Edition network; everything supported on the local network is also supported on the remote network. Pass the URL of the remote Oracle Blockchain Platform instance (`-r`), user name (`-u`) and password (`-s`) options to the command.

### Example

```
ochain invoke createSupplier
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}' -r
[https://%3cblockchain-instance-url%3e:7443/]https://<blockchain-
instance-url>:7443/
-u idcqa -s password
```

#### Testing Token Projects on a Remote Oracle Blockchain Platform Network

You can test chaincode projects that work with tokens by using Blockchain App Builder, the Oracle Blockchain Platform REST proxy, or the Hyperledger Fabric SDK.

#### Blockchain App Builder

You can use the Blockchain App Builder CLI to invoke transactions with multiple users to test token chaincodes.

To test with multiple users, change the authorization parameters (user name and password options) in the invoke and query commands.

#### Oracle Blockchain Platform REST Proxy

You can use the REST proxy in Oracle Blockchain Platform to run your token chaincode on a remote Oracle Blockchain Platform network. Use any REST Proxy client, such as Postman REST Client, to test your chaincode project.

To test multiple users, change the authorization parameters (user name and password) in your REST client, or connect to a different instance of Oracle Blockchain Platform.

## Execute Berkeley DB SQL Rich Queries

If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

If you have used `executeQuery` in the `customMethods` section of the specification file, a corresponding `executeQuery` method will be created in the controller.

Specification file:

```
customMethods:
    - executeQuery
    - "fetchRawMaterial(supplierid: string, rawMaterialSupply: number)"
    - "getRawMaterialFromSupplier(manufacturerId: string, supplierld:
string, rawMaterialSupply: number)"
    - "createProducts(manufacturerId: string, rawMaterialConsumed: number,
productsCreated: number)"
    - "sendProductsToDistribution()"
```

Controller file:

```
**
*
* BDB sql rich queries can be executed in OBP CS/EE.
* This method can be invoked only when connected to remote OBP CS/EE network.
*
*/
@Validator(yup.string())
public async executeQuery(query: string) {
    const result = await this.query(query);
    return result;
}
```

You can invoke this method to execute Berkeley DB SQL rich queries on Oracle Blockchain Platform network, ensuring that you use the `-r`, `-u` and `-s` options to specify the remote Oracle Blockchain Platform instance URL, user name and password respectively.

Example query for TypeScript:

```
ochain query  executeQuery "SELECT key, valueJson FROM <STATE> WHERE
json_extract(valueJson, '$.rawMaterialAvailable') = 4" -r
[https://%3cblockchain-instance-url%3e:7443/]https://<blockchain-
instance-url>:7443/
-u idcqa -s password
```

Example query for Go:

```
ochain query  executeQuery "SELECT key, valueJson FROM <STATE> WHERE
json_extract(valueJson, \"$.rawMaterialAvailable\") = 4" -r
[https://%3cblockchain-instance-url%3e:7443/]https://<blockchain-
instance-url>:7443/
-u idcqa -s password
```

The entire SQL query is taken in the argument, so you can make changes to your query on the fly. The syntax is different for TypeScript and Go chaincode. As shown in the previous example, the Go query uses double quotation marks for the query parameters instead of single quotation marks. The double quotation marks must be preceded by backslash characters.

## Upgrading Chaincode Projects in the CLI

You can use this command to upgrade existing chaincode projects to use the new features of the updated Blockchain App Builder. This command works with both TypeScript and Go projects.

For Go projects, upgrade to Go v1.20.10 before you run the command to upgrade your chaincode project.

Usage:

```
upgrade [options]
```

```
my-mac:Project myname$ ochain upgrade -h
Usage: upgrade [options]
Upgrade App Builder chaincode project
Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -p, --project <path>    (optional) Path to Chaincode project to
upgrade. If not specified, it defaults to current directory.
    -cc, --chaincode        (optional) To upgrade chaincode project
Examples :
$> ochain upgrade --project <path of chaincode dir>
```

To upgrade a chaincode project, navigate to the directory containing the project and then enter the following command:

```
ochain upgrade
```

If you encounter problems while upgrading a chaincode project, you can use the `--debug` option to generate debug logs. On Microsoft Windows, use Command Prompt instead of PowerShell if you specify the `--debug` option.

After you upgrade a chaincode project, synchronize the specification file with the generated source code. For more information, see Synchronize Specification File Changes With Generated Source Code.

## Synchronize Specification File Changes With Generated Source Code

You can use the `ochain sync` command to bring new changes from the specification file to the current source files (model and controller). The command works with both TypeScript and Go projects.

Note:

- Synchronization is unidirectional: you can bring changes from your specification file into your chaincode project, but not the other way around. Changes made in your chaincode project remain as-is after the synchronizing process.
- The `ochain sync` command works only if the chaincode project was scaffolded by using a specification file. Do not delete, rename or move the specification file if you plan to synchronize any changes from the specification file to the source code in the future.
- If you used a single specification file to generate more than one chaincode project, you can synchronize only one project at a time by using the `ochain sync` command.

Usage:

```
sync [options] [...args]
```

```
my-mac:TsProject myname@ ochain sync -h
Usage: sync [options] [...args]
Synchronize Changes from spec file to the required chaincode.
Arguments:
    [...args] (optional) Sync Arguments.
Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -p, --project <path>    (optional) Path to Chaincode project to sync. If
not specified, it defaults to current directory
    -c, --confirm <bool>    (optional) Parameter to ensure if you have
resolved all the conflicts, and commit changes
Examples :
$> ochain sync
without chaincode initial arguments
```

The `ochain sync` command has two optional arguments:

- `-p / --project`

This option takes the chaincode project directory where the synchronization needs to be performed. If not specified, it defaults to the current directory.

- `-c / --confirm`
  This option takes Boolean (true/false) values. If there are any conflicts during the merging process, you must resolve those conflicts manually and set this option to true in the next synchronization cycle. Don't use this option if you're not sure that the conflicts have been merged.

If the `ochain sync` command fails and you have installed and configured all prerequisites including Git and the Git username and password, complete the following steps to reinitialize the synchronization operation:

1. Check the specification file for errors and fix any errors that you find.
2. In the chaincode project folder, remove the following folders if they are present: `.sync_temp`, `.sync_backup`, and `.sync_repo`.
3. In the `src` folder, remove the `.git` folder if it is present.
4. In the project root folder, edit the `ochain.json` file and set the `syncEnabled` property to `false`. Save and close the file.
5. Retry the `ochain sync` command.

# Apply a Patch to the Blockchain App Builder CLI

You can use the `patch` command to apply a patch to the Blockchain App Builder command-line interface (CLI).

Usage:

```
patch [options]

my-mac:TsProject myname@ ochain patch -h
Usage: patch [options]

Apply PatchFix to ochain

Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -p, --path <path>       Path to Patch Zip file
```

To apply a patch to a project, navigate to the directory that contains the project and then enter the following command. You must pass the path to the archive file that contains the patch as an option to the `patch` command.

```
ochain patch --path path_to_archive_file
```

# Writing Unit Test Cases and Coverage Reports for the Chaincode Project

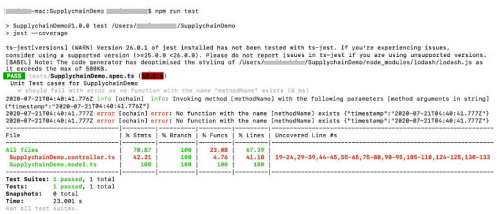
Blockchain App Builder includes support for writing unit test cases and coverage reports for the generated chaincode projects.

> **Note:**
>
> If you're running your unit tests in VS Code, it can be done in the Terminal window.

### TypeScript

To write unit test cases for a TypeScript chaincode, refer to the file `<chaincodeName>.spec.ts` in the `tests` folder inside the generated chaincode project. This file provides the complete unit testing setup for TypeScript chaincodes, and also an example unit test case in the comments section for reference. Following the example, you will be able write unit test cases for your chaincode methods.

The unit test cases can be run by executing the command `npm run test` from the chaincode project folder. This will also give you the coverage reports.



### Go

To write unit test cases for a Go chaincode, refer to the file `src/src_test.go` inside the generated chaincode project. This file provides the complete unit testing setup for Go chaincodes, and also an example unit test case in the comments section for reference. Following the example, you will be able write unit test cases for your chaincode methods.

The unit test cases can be run by executing the command `go test` from the chaincode project folder. For coverage, add the flag `--cover`.

Example: `go test --cover`.

## Generate a Postman Collection Using the CLI

You can use the `generate` command to create a Postman collection that includes example payloads for all of your chaincode controller APIs.

Postman is a tool that you can use to work with and test REST APIs. The generate command creates a Postman collection that is based on the chaincode that was automatically generated from a declarative specification file. The Postman collection contains the payloads for all of the methods that are specified in the chaincode controller file. You can change the variable values in the Postman collection file to make REST API calls.

The generated Postman collection includes default values for all APIs in the controller. To learn more about Postman, see https://www.postman.com/. After you generate a Postman collection, you can directly import it and use it by changing the default values in the payload and variables.

Usage:

```
generate [options]

my-mac:TsProject myname$ ochain generate -h
Usage: generate [options]

Generates the postman collection for the chaincode.

Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -c, --collection        This option is mandatory to generate a
Postman collection.
    -p, --project <path>    Path to the chaincode project to generate
the Postman collection from. If not specified, it defaults to current
directory.
    -o, --out <path>        Path to the generated Postman collection
JSON file. If not specified, it defaults to current directory.
```

To generate a Postman collection, navigate to the directory that contains the project and then enter the following command. You must run the `generate` command from the chaincode directory or an error will occur. If the specified Postman collection already exists, you are prompted whether to overwrite it.

`ochain generate --collection --project` *`path_to_chaincode_project`* `--out` *`path_to_postman_collection_to_generate`*

### Postman Collection Structure

The generated Postman collection includes two types of requests, invoke requests and query requests:

- Invoke requests include all write operations, which use the endpoint `/transactions`
- Query requests include all get operations, which use the endpoint `/chaincode-queries`

To differentiate between getter and non-getter methods in the controller APIs, a decorator is used in TypeScript chaincodes and a comment is used in Go chaincodes. If you define a getter method in the controller, you must use the GetMethod decorator for TypeScript or the GetMethod comment for Go, as shown in the following table.

| TypeScript | Go |
|---|---|
| Every getter method has a `GetMethod` decorator:<br><br>`@GetMethod()`<br>`@Validator()`<br>`public async getAllTokenAdmins() {`<br>`  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ADMIN.getAllAdmins", "TOKEN");`<br>`  return await this.Ctx.ERC1155Admin.getAllAdmins();`<br>`}` | Every getter method has a `GetMethod` comment block:<br><br>`/**`<br>` * GetMethod`<br>` */`<br>`func (t *Controller) GetAllTokenAdmins() (interface{}, error) {`<br>`  auth, err := t.Ctx.Auth.CheckAuthorization("Admin.GetAllAdmins", "TOKEN")`<br>`  if err != nil && !auth {`<br>`  return nil, fmt.Errorf("error in authorizing the caller  %s", err.Error())`<br>`  }`<br>`  return t.Ctx.Admin.GetAllTokenAdmins()`<br>`}` |

Generated Postman collections include variables with default values, as shown in the following table.

| Variable Name | Description | Default Value | Context |
|---|---|---|---|
| `bc-url` | The REST proxy URL of the Oracle Blockchain Platform instance where the chaincode is deployed | `https://test-xyz-abc.blockchain.ocp.oraclecloud.com:7443/restproxy` | all chaincodes |
| `bc-channel` | The channel where the chaincode is deployed | `default` | all chaincodes |
| `bc-admin-user` | The name of the admin user (a user with the admin role that can access all POST requests). By default, this user is the caller of all POST requests in the chaincode | `bc-admin-user value` | all chaincodes |
| `bc-admin-password` | The password for the admin user | `bc-admin-password value` | all chaincodes |
| `bc-timeout` | The timeout value in the body of every POST request to indicate the timeout interval | `6000` | all chaincodes |

| Variable Name | Description | Default Value | Context |
|---|---|---|---|
| bc-sync | The sync value in the body of every POST request to indicate whether the request is synchronous or asynchronous | true | all chaincodes |
| bc-chaincode-name | The chaincode name, which is used in every POST request | chaincode name | all chaincodes |
| bc-org-id | The default orgId parameter for all POST requests | bc-org-id value | token chaincodes only |
| bc-user-id | The default userId parameter for all POST requests | bc-user-id value | token chaincodes only |
| bc-token-id | The default tokenId parameter for all POST requests | bc-token-id value | token chaincodes only |

In every generated request, all of the parameters with default values are generated. Functions that have struct/class parameters will have a placeholder object in the request body, as shown in the following examples.

**API with a struct/class parameter**

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "CreateArtCollectionToken",
        "{\"TokenId\":\"{{bc-token-id}}\",\"TokenDesc\":\"TokenDesc
value\",\"TokenUri\":\"TokenUri value\",\"TokenMetadata\":
{\"Painting_name\":\"Painting_name
value\",\"Description\":\"Description value\",\"Image\":\"Image
value\",\"Painter_name\":\"Painter_name
value\"},\"Price\":999,\"On_sale_flag\":true}",
        "quantity value"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

**API without a struct/class parameter**

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "CreateAccount",
        "{{bc-org-id}}",
        "example_minter",
        "true",
```

```
        "true"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

The default value for most API parameters is *parameter_name* value, with some exceptions. The following examples show some of the exceptions.

- The filters parameter in `GetAccountTransactionHistoryWithFilters`:

```
"{\"PageSize\":20,\"Bookmark\":\"\",\"StartTime\":\"2022-01-16T15:16:36+00
:00\",\"EndTime\":\"2022-01-17T15:16:36+00:00\"}"
```

- The filters parameter in `GetSubTransactionsByIdWithFilters`:

```
"{\"PageSize\":20,\"Bookmark\":\"\"}"
```

A struct or class has different default values, as shown in the following table:

| Data Type | Default Value |
|---|---|
| boolean/bool | true |
| int/number | 999 |
| date | 2022-01-16T15:16:36+00:00 |
| other | *parameter_name* value |

### ERC-1155 Token Projects

The ERC-1155 standard includes common methods for both fungible and non-fungible tokens. The generated Postman collection for an ERC-1155 project that uses both fungible and non-fungible tokens includes two different POST requests, one for each type of token, for these common methods. If an ERC-1155 project uses only fungible or non-fungible tokens but not both types, then the generated Postman collection includes only one POST request for these common methods. The following table illustrates the generated API for the `AddRole` method.

| | Fungible Tokens | Non-Fungible Tokens |
|---|---|---|
| Request Name | `AddRole -For Fungible` | `AddRole -For NonFungible` |

| | Fungible Tokens | Non-Fungible Tokens |
|---|---|---|
| Request Body | { "chaincode": "{{bc-chaincode-name}}", "args": [ "AddRole", "{{bc-org-id}}", "{{bc-user-id}}", "role value (for example, minter / burner)", "{\"TokenId\":\"{{bc-token-id}}\"}" ], "timeout": {{bc-timeout}}, "sync": {{bc-sync}} } | { "chaincode": "{{bc-chaincode-name}}", "args": [ "AddRole", "{{bc-org-id}}", "{{bc-user-id}}", "role value (for example, minter / burner)", "{\"TokenName\":\"TokenName value\"}" ], "timeout": {{bc-timeout}}, "sync": {{bc-sync}} } |

# Troubleshoot Blockchain App Builder CLI

The following information can be used to troubleshoot system problems with Blockchain App Builder CLI.

**Prerequisites issues**
Errors can occur if you modify or upgrade any of the prerequisite software that is required by Blockchain App Builder. You can use the `preReqCheck` command to check that your installation of Blockchain App Builder still meets the prerequisites.
Usage:

```
preReqCheck [options]

my-mac:TsProject myname$ ochain preReqCheck -h
Usage: patch [options]

Validates the pre-requisites of Blockchain App Builder

Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
```

If the prerequisites check fails with errors and warnings when you attempt to install Blockchain App Builder, you might see an error similar to the following example:

```
npm ERR! code 1
npm ERR! path
```

```
C:\Users\opc\AppData\Roaming\npm\node_modules\@oracle\ochain-cli
npm ERR! command failed
npm ERR! command C:\Windows\system32\cmd.exe /d /s /c node build/pre-
install.js
npm ERR! NodeJs version is correct.
npm ERR! NPM version is correct.
npm ERR! Error:
npm ERR! Found 1 error(s) in pre-requisites check, failed with following
errors:
npm ERR! 1. Golang version mismatch. Expected 1.20.x, but found 1.18.5.
npm ERR!
npm ERR! Found 3 warning(s) in pre-requisites check.
npm ERR! 1. Docker is not installed. Please install Docker >= 18.09.0. To
deploy chaincodes in the local environment, please install the recommended
version of Docker.
npm ERR! 2. Docker Compose is not installed. Please install Docker Compose
>= 1.23.0. To deploy chaincodes in the local environment, please install the
recommended version of Docker Compose.
npm ERR! 3. Git is not Installed. To sync chaincodes, please install the Git
according to the documentation.
npm ERR!     at preRequsiteCheck
(C:\Users\opc\AppData\Roaming\npm\node_modules\@oracle\ochain-
cli\build\lib\util\pre-install.js:435:15)
npm ERR!     at Object.<anonymous>
(C:\Users\opc\AppData\Roaming\npm\node_modules\@oracle\ochain-cli\build\pre-
install.js:10:44)
npm ERR!     at Module._compile (node:internal/modules/cjs/loader:1254:14)
npm ERR!     at Module._extensions..js (node:internal/modules/cjs/
loader:1308:10)
npm ERR!     at Module.load (node:internal/modules/cjs/loader:1117:32)
npm ERR!     at Module._load (node:internal/modules/cjs/loader:958:12)
npm ERR!     at Function.executeUserEntryPoint [as runMain] (node:internal/
modules/run_main:81:12)
npm ERR!     at node:internal/main/run_main_module:23:47

npm ERR! A complete log of this run can be found in:
npm ERR!     C:\Users\opc\AppData\Local\npm-
cache\_logs\2023-08-25T09_58_34_514Z-debug-0.log
```

**Deployment failure**

Due to deployment failure, corrupt deployment, a Docker peer container being full, or a Docker peer being killed in the local network, you may see an error similar to:

```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating Chaincode
GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container: Failed to
generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error (404):
manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
```

```
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting
container: Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API
error (404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown
exited: signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again. To work around this behavior, complete the following step.

- Open a new terminal window in the chaincode project and run the `ochain run -b` command.

Blockchain App Builder CLI rebuilds the local fabric environment and deploys your chaincode to the new environment.

**Mac OSX: Xcode**
After a Mac OSX upgrade, or if Xcode is not installed, you might see an error similar to the following in the error log:

```
gyp: No Xcode or CLT version detected!
gyp ERR! configure error
gyp ERR! stack Error: `gyp` failed with exit code: 1
gyp ERR! stack     at
```

- To work around this behavior, open a terminal window and run the following commands:

  ```
  sudo rm -rf $(xcode-select --print-path)
  xcode-select --install
  ```

# Using the Blockchain App Builder Extension for Visual Studio Code

The Blockchain App Builder extension for Visual Studio Code helps you build and scaffold a fully-functional chaincode project from a specification file.

After the project is built, you can run and test it on a local Hyperledger Fabric network, or your provisioned Oracle Blockchain Platform network. You can then run SQL rich queries, debug the chaincode, or write and run unit tests using the generated code.

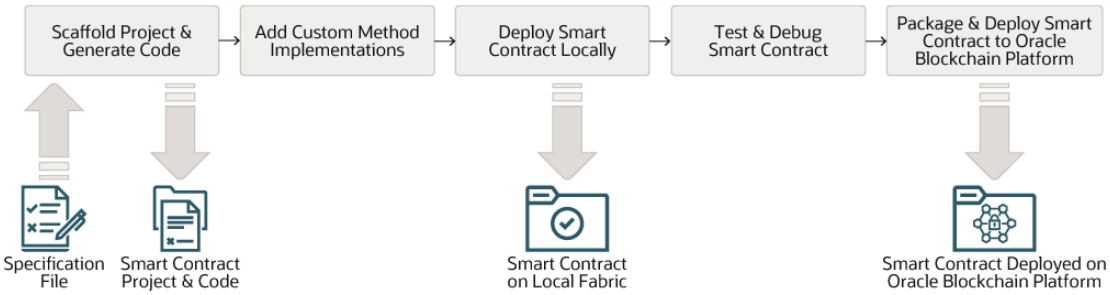



**Table 7-3 Workflow When Using the VS Code Extension**

| Task | Description | More Information |
|---|---|---|
| Install and configure | Download the Blockchain App Builder VS Code extension from your Oracle Blockchain Platform console and install it and any prerequisite software. | • Install and Configure the Blockchain App Builder Extension for Visual Studio Code |
| Create the chaincode project | Create a specification file for the chaincode project. | • Create a Chaincode Project with the Blockchain App Builder VS Code Extension |
| Generate the chaincode | Edit the specification file to define the assets and chaincodes to generate, and then generate your chaincode from the specification file. | Detailed reference information about the structure and contents of the specification file and the generated chaincode project:<br>• Input Specification File<br>• Scaffolded TypeScript Chaincode Project<br>• Scaffolded Go Chaincode Project<br>Detailed information about tokenization support:<br>• Tokenization Support Using Blockchain App Builder<br>• Scaffolded TypeScript NFT Project for ERC-721<br>• Scaffolded Go NFT Project for ERC-721<br>• Scaffolded TypeScript Project for Token Taxonomy Framework<br>• Scaffolded Go Project for Token Taxonomy Framework |
| Deploy the chaincode | After your chaincode project is created, you can deploy it locally to the included pre-configured Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition.<br>You can also package the chaincode project for manual deployment to Oracle Blockchain Platform. | • Deploy the Chaincode to a Local Hyperledger Fabric Network<br>• Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network<br>• Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform |

**Table 7-3 (Cont.) Workflow When Using the VS Code Extension**

| Task | Description | More Information |
|---|---|---|
| Test the chaincode | After your chaincode is running on a network, you can test any of the generated methods.<br>Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network. | • Test Your Chaincode on a Local Hyperledger Fabric Network<br>• Testing Lifecycle Operations on a Remote Oracle Blockchain Platform Network<br>• Execute Berkeley DB SQL Rich Queries |
| Debug the chaincode | You can do line-by-line debugging in Visual Studio Code. | • Debugging from Visual Studio Code |
| Synchronize your updates | When you update your specification file, you can synchronize the changes with the generated chaincode files. | • Synchronize Specification File Changes With Generated Source Code |
| Running unit tests | A basic unit test case setup is included in the project. Additional tests can be added and run. | • Writing Unit Test Cases and Coverage Reports for the Chaincode Project |

# Install and Configure the Blockchain App Builder Extension for Visual Studio Code

The Blockchain App Builder extension for Visual Studio Code can be downloaded through the Oracle Blockchain Platform console.

The following platforms are supported:

- macOS
- Oracle Linux 8.0 or 9.0
- Microsoft Windows 10 or 11

### Prerequisites

Before you install Blockchain App Builder on your local system, you must install the prerequisites.

> **Note:**
>
> Blockchain App Builder coordinates with Oracle Blockchain Platform and its compilers. If you use any versions of the prerequisites other than the ones mentioned in the following section, deploying your chaincode to a remote Oracle Blockchain Platform network might fail.

When you install Blockchain App Builder, a prerequisites check runs first. If the prerequisites check fails, the installation process is stopped.

- macOS
- Linux
- Windows

## macOS

**Prerequisites**

- Rancher Desktop (tested with 1.4.1). Blockchain App Builder can also work with Docker, but it has been tested and verified with Rancher Desktop. If you plan to use Rancher Desktop, uninstall Docker completely before installing Rancher Desktop. After you install Rancher Desktop, ensure that the container runtime is set to `dockerd (moby)`. To verify the container runtime in Rancher Desktop 1.4.1, click **Kubernetes Settings**, and then **Container Runtime**.
- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0), and npm v8.x or 9.x (tested with 9.5.0 and 9.5.1)
  Check the Node.js version by running the following command: `node --version`

  Check the npm version by running the following command: `npm --version`

  If you use a manager such as `nvm` or `nodenv` to install Node.js and npm, set the default/global version and then restart Visual Studio Code so that the version will be detected by the Prerequisites page.

  Do not use versions of Node.js earlier or later than version 18.
- Go version v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
  Check the Go version by running the following command: `go version`
- If you plan to use the synchronization feature of Blockchain App Builder, install Git and configure your user name and email as shown in the following commands. Specify your user name and email address in the place of `<your_name>` and `<email>`.

  ```
  git config --global user.name "<your_name>"


  git config --global user.email "<email>"
  ```

- Visual Studio Code version 1.66.0 or later
  Check the Visual Studio Code version by running the following command: `code --version`

**Install Node.js and npm by Using nvm**

Using nvm to install Node.js and npm gives you the ability to run more commands without `sudo`.

1. Enter the following command to install nvm:

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh
   | bash
   ```

2. Add the following code snippet to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
# This loads nvm bash_completion
```

3. Log out and then log back in to your operating system.
4. Enter the following command to verify the nvm installation:

```
nvm version
```

5. Enter the following command to install Node.js and npm:

```
nvm install 18.16.0
```

6. Enter the following command to set Node.js 18.16.0 as the default in nvm:

```
nvm alias default 18.16.0
```

The output of the command is the following text:

```
default -> 18.16.0 (-> v18.16.0)
```

## Linux

**Prerequisites**

- Docker v18.09.0 or later
- Docker Compose v1.23.0 or later
- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0), and npm v8.x or v9.x (tested with 9.5.0 and 9.5.1)
  Check the Node.js version by running the following command: `node --version`

  Check the npm version by running the following command: `npm --version`

  If you use a manager such as `nvm` or `nodenv` to install Node.js and npm, set the default/global version and then restart Visual Studio Code so that the version will be detected by the Prerequisites page.

  Do not use versions of Node.js earlier or later than version 18.
- Go version v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
  Check the Go version by running the following command: `go version`
- If you plan to use the synchronization feature of Blockchain App Builder, install Git and configure your user name and email as shown in the following commands.

Specify your user name and email address in the place of `<your_name>` and `<email>`.

```
git config --global user.name "<your_name>"


git config --global user.email "<email>"
```

- Visual Studio Code version 1.66.0 or later
  Check the Visual Studio Code version by running the following command: `code --version`

**Install Node.js and npm by Using nvm**

Using nvm to install Node.js and npm gives you the ability to run more commands without `sudo`.

1. Enter the following command to install nvm:

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh
   | bash
   ```

2. Add the following code snippet to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

   ```
   export NVM_DIR="$HOME/.nvm"
   [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
   [ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  #
   This loads nvm bash_completion
   ```

3. Log out and then log back in to your operating system.
4. Enter the following command to verify the nvm installation:

   ```
   nvm version
   ```

5. Enter the following command to install Node.js and npm:

   ```
   nvm install 18.16.0
   ```

6. Enter the following command to set Node.js 18.16.0 as the default in nvm:

   ```
   nvm alias default 18.16.0
   ```

   The output of the command is the following text:

   ```
   default -> 18.16.0 (-> v18.16.0)
   ```

## Windows

**Prerequisites**

- Rancher Desktop (tested with 1.4.1). Blockchain App Builder can also work with Docker, but it has been tested and verified with Rancher Desktop.

- The latest release of Node.js version 18 (tested with 18.15.0 and 18.16.0). Do not use versions of Node.js earlier or later than version 18.
- npm v8.x or v9.x (tested with 9.5.0 and 9.5.1)
- Go v1.20.10. After installing Blockchain App Builder, see Additional Setup for Go Chaincode Projects.
- If you want to use the synchronization feature of Blockchain App Builder, install Git and configure your user name and email as shown in the following commands. Specify your user name and email address in the place of `<your_name>` and `<email>`.

```
git config --global user.name "<your_name>"


git config --global user.email "<email>"
```

### Install Rancher Desktop

Complete the following steps to install Rancher Desktop on Microsoft Windows.

1. If Docker is installed on your local computer, uninstall it completely.
2. Download and install Rancher Desktop.
3. After the installation wizard completes, before you open Rancher Desktop, run the following commands:

```
wsl --install
wsl --set-default-version 2
wsl --setdefault rancher-desktop
```

4. Open Rancher Desktop to complete the setup process.
5. After you install Rancher Desktop, ensure that the container runtime is set to `dockerd (moby)`. To verify the container runtime in Rancher Desktop 1.4.1, click **Kubernetes Settings**, and then **Container Runtime**.

---

### Install the Blockchain App Builder Extension

1. Download the extension from the **Developer Tools** tab on the **Blockchain App Builder** pane of the Oracle Blockchain Platform console. On the **Blockchain App Builder** pane, under the **Download** section, select **Visual Studio Code Extension**.
2. In Visual Studio Code, open the **Extensions** panel and then from the **More Actions** menu, select **Install from VSIX**.
3. Locate the downloaded `oracle-ochain-extension-x.x.x.vsix` file and then click **Install**. (Adjust the name of the `.vsix` file for the version that you are installing.)
4. Restart Visual Studio Code to complete installation of the extension.
5. To use the specification file validation functions, which automatically validate the specification file as you type, install the YAML extension from Red Hat. Open the

**Extensions** panel, search for YAML, install the YAML Language Support extension, and then restart Visual Studio Code.

After installation, you can use the Oracle Blockchain App Builder icon on the left side of Visual Studio Code to open the Blockchain App Builder panel.

Additionally, the Blockchain App Builder command line interface (CLI) is automatically installed as part of the extension for Visual Studio Code if you haven't already installed it separately. The CLI commands can be run from any terminal application, including the Visual Studio Code console window. Blockchain App Builder is installed globally, so you can run the CLI commands from any location in the file system.

### Additional Setup for Go Chaincode Projects

To develop a Go project, you must set the `GOPATH` environment variable. This allows Go to locate your workspace and run your code.

- macOS
- Linux
- Windows

## macOS

Before setting the `GOPATH` environment variable, make sure that a `go/` folder exists in your `$HOME` directory. If not, enter the following command to create a `go/` directory in your home directory:

```
mkdir $HOME/go
```

Set your `GOPATH` environment variable by adding the following variables to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

After editing the file, run the following command to make your changes take effect immediately:

```
source ~/.bash_profile
```

Alternately, you can apply the change system-wide by adding the previous variables to the `/etc/bashrc` file.

### Linux

Before setting the `GOPATH` environment variable, make sure that a `go/` folder exists in your `$HOME` directory. If not, enter the following command to create a `go/` directory in your home directory:

```
mkdir $HOME/go
```

Set your `GOPATH` environment variable by adding the following variables to the applicable file: `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

After editing the file, run the following command to make your changes take effect immediately:

```
source ~/.bash_profile
```

Alternately, you can apply the change system-wide by adding the previous variables to the `/etc/bashrc` file.

### Windows

Create a `go/` directory in your home directory: `C:\Users\<username>\go`.

## Upgrade the Blockchain App Builder Extension for Visual Studio Code

To upgrade Blockchain App Builder, you must uninstall the previous version and then install the newer version.

1. In Visual Studio Code, open the **Extensions** panel and then select the Blockchain App Builder extension.
   The Blockchain App Builder extension page is displayed.
2. Click **Uninstall**.
3. After Blockchain App Builder is uninstalled, restart Visual Studio Code.
4. Download the latest version of the Blockchain App Builder extension from the **Developer Tools** tab on the Blockchain App Builder pane of the Oracle Blockchain Platform console, and then install Blockchain App Builder. For more information, see Install and Configure the Blockchain App Builder Extension for Visual Studio Code.

# Create a Chaincode Project with the Blockchain App Builder VS Code Extension

To create a Chaincode Project when using the Blockchain App Builder, you need to scaffold a chaincode project from a detailed specification file. This generates a project with all the files you need.

### Background

Blockchain App Builder initializes and scaffolds a chaincode project right out of the box for you. Based on simple input, **Create New Chaincode** can generate complex chaincode projects with features such as:

- Multiple assets (models) and their behaviors (controllers)
- Auto-generate CRUD (Create/Read/Update/Delete) and non-CRUD methods
- Automatic validation of arguments
- Marshalling/unmarshalling of arguments
- Transparent persistence capability (ORM)
- Calling rich queries
- Transient and private data support
- Identity management

The generated project follows model/controller and decorator pattern, which allows an asset's properties maintained on the ledger to be specified as typed fields and extended with specific behaviors and validation rules. This reduces the number of lines of code which helps in readability and scalability.

### Create a Specification File

Before you begin, you need to create an input specification file. Note that you cannot alter the sample specification files that were installed as part of Blockchain App Builder, but you can duplicate them or use them as a reference file for your own specification files.

1. In the **Specifications** pane, select **Create New Spec File**.
2. The **Specifications Details** pane opens:
   - Enter the name for the specification file.
   - Select the file type - YAML and JSON are supported.
   - Optionally enter a description for the file.
   - The **Reference File** drop-down allows you to generate your specification file from a pre-existing file in your workspace if you have a file you'd like to use as a template. If nothing is selected, the created file will be empty and you can enter your specification from scratch.
   - Enter the location where you want the specification file to be stored on your system.

   Click **Save**.

The new specification file is created and appears in the **Specifications** pane. Click on it to open it in the editor.

#### Import a Specification File

If you have a pre-existing specification file, you can import it:

1. In the **Specifications** pane, click **More Actions** and select **Import Specification**.
2. Browse to your file and click **Import Specification**.

The specification file is imported and appears in the **Specifications** pane. Click on it to open it in the editor.

#### Duplicate a Specification File

You can also duplicate a specification file that's already in your **Specifications** pane by right-clicking it and selecting **Duplicate**. Right-click the file and select **Rename** to update the name.

The details about the contents of specifications files are described here: Input Specification File. Use this information and the sample specification templates to create your specification content.

#### Validate a Specification File

If you installed the YAML Language Support extension, the specification file is automatically validated as you type. The following validation features are supported automatically:

- Command completion
- Completion of scalar nodes to schema defaults
- Context-sensitive help when you hover over a property or method
- Indents for array items as you type
- Syntax error detection

#### Scaffolding the Chaincode Project

When you have a specification file that meets your needs, generate your chaincode project.

1. In the **Chaincodes** pane, select **Create New Chaincode**.
2. The **Chaincode Details** pane opens:
   - Enter the name of your chaincode project.
   - Select the language: TypeScript or Go.
   - Select the specification file that you're using to create the chaincode.
   - Enter the location or Go domain where you want the project to be created within your local development environment.

   Click **Create**.

When your project is created, it will be shown in the **Chaincodes** pane. All the files required by the chaincode will be in the project. For a detailed overview of the files created, see:

- Scaffolded TypeScript Chaincode Project
- Scaffolded Go Chaincode Project

For a detailed overview of a token-based project, see also:

- Scaffolded TypeScript Token Project for ERC-1155
- Scaffolded Go Token Project for ERC-1155
- Scaffolded TypeScript NFT Project for ERC-721
- Scaffolded Go NFT Project for ERC-721
- Scaffolded TypeScript Project for Token Taxonomy Framework
- Scaffolded Go Project for Token Taxonomy Framework

> **Note:**
>
> - The **Chaincodes** pane allows you open and work with content within the chaincode project, but won't let you add, delete, or rename files within the project. To do that, right-click your project and select **Open in Explorer**. This opens the project in the VS Code Explorer view.
> - Deleting or renaming files in the chaincode project can potentially break the link between the project files and the specification file used to create it. If you plan to synchronize your code between the two, don't change the file names.

**Import an Existing Chaincode Project**

If you've created a chaincode project through the CLI or you've cleaned your VS Code blockchain content and want to import a locally saved project, in the **Chaincodes** pane click the **More Actions...** icon and select **Import Chaincode**. Browse to the project and click **Import Chaincode**.

## Input Specification File

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

With the specification file you can specify multiple asset definitions and behavior, CRUD and non-CRUD method declaration, custom methods, validation of arguments, auto marshalling/unmarshalling, transparent persistence capability, and invoking rich data queries using SQL SELECTs or CouchDB Query Language. These features will be generated for you.

For information on specifying token assets see the following topics:

- Input Specification File for Token Taxonomy Framework
- Input Specification File for ERC-721
- Input Specification File for ERC-1155

The specification file can be written in either `yaml` or `json`. You can see sample specification files in both formats in the Blockchain App Builder package download:

- `Fabcar-Typescript.yml`
- `Marbles-Go.yml`

> **Note:**
>
> As per Go conventions, exported names begin with a capital letter. Therefore all the asset properties and methods must have names starting with capital letters in the specification file.

### Structure of the Specification File

Typically, you structure a specification file in the following way:

```
assets:
    name:
    type:
    properties:
        name:
        type:
        id:
        derived:
           strategy:
           algorithm:
           format:
        mandatory:
        default:
        validate:
    methods:
        crud:
        others:
customMethods:
```

Blockchain App Builder supports two special asset types, embedded assets and token assets, in addition to generic assets with no specified type. The special assets are defined as `type: embedded` or `type: token` under the `assets:` section of the specification file.

**Table 7-4 Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`assets:`** | This property takes the definition and behavior of the asset. You can give multiple asset definitions here. | |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`name:`** | The name of the asset.<br>The following names are reserved. Do not use these names for assets.<br>• `account`<br>• `role`<br>• `hold`<br>• `token`<br>• `authorization`<br>• `tokenAdmin`<br>• `Account`<br>• `Role`<br>• `Hold`<br>• `Token`<br>• `Authorization`<br>• `TokenAdmin` | `name: owner # Information about the owner` |
| **`type:`** | **Asset types**<br>The following special asset types are supported:<br>• `embedded`<br>• `token`<br>If you do not specify a `type` parameter in the `assets` section, the asset is of the generic type. | |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`type:`**<br>*type: embedded* | If this property is set to `embedded` the asset is defined as an embedded asset. Embedded assets do not have CRUD methods and have to be part of another asset to store in the ledger.<br>In the example, the property `address` is embedded, and is defined in another asset.<br>Embedded assets do not support circular references. For instance, in the previous example the `address` asset cannot contain a reference to the `employee` asset. | Asset: `employee`<br><br>`name: employee`<br>`properties:`<br>`name: employeeId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br><br>`name: firstName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br><br>`name: lastName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br><br>`name: age`<br>`type: number`<br>`validate: positive(),min(18)`<br><br>**`name: address`**<br>**`type: address`**<br><br>Asset: `address`<br><br>**`name: address`**<br><br>**`type: embedded`**<br><br>`properties:`<br>`name: street`<br>`type: string`<br><br>`name: city`<br>`type: string`<br><br>`name: state`<br>`type: string`<br><br>`name: country`<br>`type: string` |
| **`properties:`** | Describe all the properties of an asset. | |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`name:`** | The name of the property. | `name: ownerId # Unique ID for each owner` |
| **`id:`** | • true<br>This specifies the identifier of this asset. This property is mandatory. | `name: owner # Information about the owner`<br>`properties:`<br>`  name: ownerId # Unique ID for each owner`<br>`  type: string`<br>`  mandatory: true`<br>**`  id: true`**<br>`  name: name # Name of the owner`<br>`  type: string`<br>`  mandatory: true` |
| **`type:`** | **Property types**<br>The following basic property types are supported:<br>• `number`<br>• `float`<br>• `string`<br>• `boolean`<br>• `date`<br>• `array`<br>For Go chaincodes, `number` is mapped to `int` and `float` is mapped to `float64`. Other types are not currently supported, including the following types:<br>• `complex`<br>• `unsigned/ signed int`<br>• `8/16/32/64 bits` | `name: year # Model year`<br>**`  type: number`**<br>`  mandatory: true`<br>`  validate: min(1910),max(2020)`<br>`  name: color # Color - no validation as color names are innumerable`<br>**`  type: string`**<br>`  mandatory: true` |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **derived:** | This property specifies that the id property is derived from other keys. Dependent properties should be `string` datatype and not an embedded asset.<br>This property has two mandatory parameters:<br>• `strategy`: takes values of `concat` or `hash`.<br>• `format`: takes an array of specification strings and values to be used by the strategy.<br>Example 1:<br>• The property `employeeID` is dependent on the `firstName` and `lastName` properties.<br>• This property is a concatenation of the values listed in the `format` array.<br>• `IND%1#%2%tIND` is the 0th index in the array and describes the final format.<br>• `%n` is a position specifier that takes its values from the other indexes in the array.<br>• `%t` indicates the value should be `stub.timestamp` from the channel header.<br>• If you need to use the character `%` in the `format` | Example 1<br>`name: employee`<br>`properties:`<br>`name: employeeId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br>**`derived:`**<br>**`strategy: concat`**<br>**`format:`**<br>**`["IND%1#%2%tIND","firstName","lastName"]`**<br>`name: firstName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: lastName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: age`<br>`type: number`<br>`validate: positive(),min(18)`<br>Example 2<br>`name: account`<br>`properties:`<br>`name: accountId`<br>`type: string`<br>`mandatory: true`<br>`id: true`<br>**`derived:`**<br>**`strategy: hash`**<br>**`algorithm: 'sha256'`**<br>**`format:`**<br>**`["IND%1#%2%t","bankName","ifsccode"]`**<br>`name: bankName`<br>`type: string`<br>`validate: max(30)`<br>`mandatory: true`<br>`name: ifsccode` |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| | string, it should be escaped with another `%`.<br>• The final format in this example would be: `INDfirstName#lastName1606885454916IND`<br>Example 2:<br>• When using `hash`, you must also use the `algorithm` parameter. The default is `sha256`; `md5` is also supported.<br>• `IND%1#%2%t` is the 0th index in the array and describes the final format.<br>• `%n` is a position specifier that takes its values from the other indexes in the array.<br>• `%t` indicates the value should be `stub.timestamp` from the channel header.<br>• If you need to use the character `%` in the `format` string, it should be escaped with another `%`. | `type: string`<br>`mandatory: true` |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`mandatory:`** | • true<br>• false<br>The corresponding property is mandatory and cannot be skipped while creating an asset. | `name: phone # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`<br>`type: string`<br>**`mandatory: true`**<br>`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`<br>`name: cars # The list of car VINs owned by this owner`<br>`type: string[]`<br>**`mandatory: false`** |
| **`default:`** | This gives you the default value of this property. | |
| **`validate:`** | The given property is validated against some of the out-of-box validations provided by Blockchain App Builder. You can chain validations if you ensure that the chain is valid.<br>If the `validate` property is not provided, then the validation is done against only the property `type`. | |
| **`validate:`**<br>*type: number* | • positive()<br>• negative()<br>• min()<br>• max()<br>These validations can be chained together separated by commas. | `name: offerApplied`<br>`type: number`<br>**`validate: negative(),min(-4)`**<br><br>`name: year # Model year`<br>`type: number`<br>`mandatory: true`<br>**`validate: min(1910),max(2020)`** |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **validate:**<br>*type: string* | • `min()`<br>• `max()`<br>• `email()`<br>• `url()`<br>• `/regex/` - supports PHP regex<br>For Go chaincodes, regular expressions which contain certain reserved characters or whitespace characters should be properly escaped. | `name: website`<br>`type: string`<br>`mandatory: false`<br>**`validate: url()`**<br><br>`name: phone  # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`<br>`type: string`<br>`mandatory: true`<br>**`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`**<br><br>`name: Color #Color can be red, blue, or green`<br>`type: string`<br>`mandatory: true`<br>**`validate: /^\\s*(red\|blue\|green)\\s*$/`** |
| **validate:**<br>*type: boolean* | • true<br>• false<br>In the example, the validation of property `active` is by the type itself (`boolean`) | `name: active`<br>**`type: boolean`** |
| **validate:**<br>*type: array* | By type itself, in the form of `type: number[]`, this conveys that the array is of type number.<br>You can enter limits to the array in the format `number[1:5]` which means minimum length is 1, maximum is 5. If either one is avoided, only min/max is considered. | `name: items`<br>**`type: number[:5]`** |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`validate:`**<br>*type: date* | • `min()`<br>• `max()`<br>Date should be one of these formats:<br>• `YYYY-MM-DD`<br>• `YYYY-MM-DDTHH:MM:SSZ`, where `T` separates the date from the time, and the `Z` indicates UTC. Timezone offsets can replace the `Z` as in -05:00 for Central Daylight Savings Time. | `name: expiryDate`<br>`type: date`<br>**`validate: max('2020-06-26')`**<br><br>`name: completionDate`<br>`type: date`<br>**`validate: min('2020-06-26T02:30:55Z')`** |
| **`methods:`** | Use this to state which of the CRUD (Create/Read/Update/Delete) or additional methods are to be generated.<br>By default, if nothing is entered, all CRUD and other methods are generated. | `methods:`<br>`  crud: [create, getById, update, delete]`<br>`  others: [getHistoryById, getByRange]` |
| **`crud:`** | • `create`<br>• `getByID` (read)<br>• `update`<br>• `delete`<br>If this array is left empty, no CRUD methods will be created.<br>If the `crud` parameter is not used at all, all four methods will be created by default.<br>The `crud` parameter is not applicable to `token` and `embedded` assets. | `methods:`<br>**`  crud: [create, getById, delete]`**<br>`  others: [] # no other methods will be created` |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **others:** | • getHistoryById<br>• getByRange<br>getHistoryById returns the history of the asset in a list.<br>getByRange returns all the assets in a given range. For more information, see getByRange (TypeScript) and GetByRange (Go).<br>If this array is left empty, no other methods will be created.<br>If the others parameter is not used at all, both methods will be created by default.<br>The others parameter is not applicable to token and embedded assets. | methods:<br>crud: [create, delete]<br>**others: [] # no other methods will be created**<br><br>methods:<br>crud: [create, getById, update, delete]<br>**others: [getHistoryById, getByRange]** |

**Table 7-4 (Cont.) Specification File Parameter Descriptions and Examples**

| Entry | Description | Examples |
|---|---|---|
| **`customMethods:`** | This property creates invokable custom method templates in the main controller file. It takes the method signature and creates the function declaration in the controller file.<br>You can provide language specific function declarations here.<br>We provide a custom method named `executeQuery`. If it's added to the specification file, it details how Berkeley DB SQL and CouchDB rich queries can be executed. This method can be invoked only when you are connected to Oracle Blockchain Platform Cloud or Enterprise Edition. | TypeScript<br>`customMethods:`<br>`- executeQuery`<br>`- "buyCar(vin: string, buyerId: string, sellerId: string, price: number, date: Date)"`<br>`- "addCar(vin: string, dealerId: string, price: number, date: Date)"`<br>Go<br>`customMethods:`<br>`- executeQuery`<br>`- "BuyCar(vin string, buyerId string, sellerId string, price int)"`<br>`- "AddCar(vin string, dealerId string, price int)"` |

## Scaffolded TypeScript Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project. The project contains automatically generated classes and functions, CRUD methods, SDK methods, automatic validation of arguments, marshalling/un-marshalling and transparent persistence capability (ORM).

If the chaincode project uses the TypeScript language, the scaffolded project contains three main files:

- `main.ts`
- `<chaincodeName>.model.ts`
- `<chaincodeName>.controller.ts`

All the necessary libraries are installed and packaged. The `tsconfig.json` file contains the necessary configuration to compile and build the TypeScript project.

The `<chaincodeName>.model.ts` file in the `model` subdirectory contains multiple asset definitions and the `<chaincodeName>.controller.ts` file in the `controller` subdirectory contains the assets behavior and CRUD methods.

The various decorators in `model.ts` and `controller.ts` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

**Reference:**

- Models
- Decorators
- ORM
- SDK Methods
- Controller
- Automatically Generated Methods
- Controller Method Details
- Custom Methods
- Init Method

### Models

Every model class extends the `OchainModel` class, which has an additional read-only property called `assetType`. This property can be used to fetch only assets of this type. Any changes to this property are ignored during the creation and updating of the asset. The property value by default is `<modelName>`.

The `OchainModel` class enforces decorator behaviors on properties of the class.

```
@Id('supplierId')
export class Supplier extends OchainModel<Supplier> {
    public readonly assetType = 'supplier';
    @Mandatory()
    @Validate(yup.string())
    public supplierId: string;
```

### Decorators

#### Class decorators

`@Id(identifier)`

This decorator identifies the property which uniquely defines the underlying asset. This property is used as a key of the record, which represents this asset in the chaincode's state. This decorator is automatically applied when a new TypeScript project is scaffolded. The 'identifier' argument of the decorator takes the value from specification file.

```
@Id('supplierId')
export class Supplier extends OchainModel{
...
}
```

#### Property decorators

Multiple property decorators can be used. The decorators are resolved in top to bottom order.

```
@Mandatory()
```

This marks the following property as mandatory so it cannot be skipped while saving to the ledger. If skipped it throws an error.

```
@Mandatory()
public supplierID: string;
```

```
@Default(param)
```

This property can have a default value. The default value in the argument (`param`) is used when the property is skipped while saving to the ledger.

```
@Default('open for business')
@Validate(yup.string())
public remarks: string;
```

```
@Validate(param)
```

The following property is validated against the schema presented in the parameter. The argument `param` takes a yup schema and many schema methods can be chained together. Many complex validations can be added. Refer to https://www.npmjs.com/package/yup for more details.

```
@Validate(yup.number().min(3))
public productsShipped: number;
```

```
@ReadOnly(param)
```

This property decorator marks the underlying property as having a read-only value. The value in the argument, for example *param*, is used when the property is saved in the ledger. Once the value is set it cannot be edited or removed.

```
@ReadOnly('digicur')
public token_name: string;
```

```
@Embedded(PropertyClass)
```

This property decorator marks the underlying property as an embeddable asset. It takes the embeddable class as a parameter. This class should extend the `EmbeddedModel` class. This is validated by the decorator.

In this example, `Employee` has a property called `address` of type `Address`, which is to be embedded with the `Employee` asset. This is denoted by the `@Embedded()` decorator.

```
export class Employee extends OchainModel<Employee> {

   public readonly assetType = 'employee';

   @Mandatory()
```

```
    @Validate(yup.string())
    public emplyeeID: string;

    @Mandatory()
    @Validate(yup.string().max(30))
    public firstName: string;

    @Mandatory()
    @Validate(yup.string().max(30))
    public lastName: string;

    @Validate(yup.number().positive().min(18))
    public age: number;

    @Embedded(Address)
    public address: Address;
}


export class Address extends EmbeddedModel<Address> {

    @Validate(yup.string())
    public street: string;

    @Validate(yup.string())
    public city: string;

    @Validate(yup.string())
    public state: string;

    @Validate(yup.string())
    public country: string;
}
```

When a new instance of the `Address` class is created, all the properties of the `Address` class are automatically validated by the `@Validate()` decorator. Note that the `Address` class does not have the `assetType` property or `@Id()` class decorator. This asset and its properties are not saved in the ledger separately but are saved along with the `Employee` asset. Embedded assets are user defined classes that function as value types. The instance of this class can only be stored in the ledger as a part of the containing object (`OchainModel` assets). All the above decorators are applied automatically based on the input file while scaffolding the project.

```
@Derived(STRATEGY, ALGORITHM, FORMAT)
```

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `STRATEGY`: takes values of `CONCAT` or `HASH`. Requires an additional parameter `ALGORITHM` if `HASH` is selected. The default algorithm is `sha256`; `md5` is also supported.

- `FORMAT`: takes an array of specification strings and values to be used by the strategy.

```
@Id('supplierID')
export class Supplier extends OchainModel<Supplier> {

   public readonly assetType = 'supplier';

   @Mandatory()
   @Derived(STRATEGY.HASH.'sha256',['IND%1IND%2','license','name'])
   @Validate(yup.string())
   public supplierID: string;

   @Validate(yup.string().min(2).max(4))
   public license: string;

   @Validate(yup.string().min(2).max(4))
   public name: string;
```

**Method decorators**

`@Validator(…params)`

This decorator is applied on methods of the main controller class. This decorator is important for parsing the arguments, validating against all the property decorators and returning a model/type object. Controller methods must have this decorator to be invokable. It takes multiple user-created models or yup schemas as parameters.

The order of the parameters must be exactly the same as the order of the arguments in the method.

In the following example, the `Supplier` model reference is passed in the parameter that corresponds to the `asset` type in the method argument. At run time, the decorator parses and converts the method argument to a JSON object, validates against the `Supplier` validators, and after successful validation converts the JSON object to a `Supplier` object and assigns it to the `asset` variable. Then the underlying method is finally called.

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}
```

In the following example, multiple asset references are passed; they correspond to the object types of the method arguments. Notice the order of the parameters.

```
@Validator(Supplier, Manufacturer)
public async createProducts(supplier: Supplier, manufacturer:
Manufacturer) {
}
```

Apart from asset references, yup schema objects can also be passed if the arguments are of basic-types. In the following example, `supplierId` and `rawMaterialSupply` are

of type `string` and `number` respectively, so the yup schema of similar type and correct order is passed to the decorator. Notice the chaining of yup schema methods.

```
@Validator(yup.string(), yup.number().positive())
public async fetchRawMaterial(supplierID: string, rawMaterialSupply: number)
{
    const supplier = await this.Ctx.Model.get(supplierID, Supplier);
    supplier.rawMaterialAvailable = supplier.rawMaterialAvailable +
rawMaterialSupply;
    return await this.Ctx.Model.update(supplier);
}
```

### ORM

Transparent Persistence Capability or simplified ORM is captured in the `Model` class of the Context (`Ctx`) object. If your model calls any of the following SDK methods, access them by using `this.Ctx.Model`.

SDK methods that implement ORM are the following methods:

- `save` – this calls the Hyperledger Fabric `putState` method
- `get` – this calls the Hyperledger Fabric `getState` method
- `update` – this calls the Hyperledger Fabric `putState` method
- `delete` – this calls the Hyperledger Fabric `deleteState` method
- `history` – this calls the Hyperledger Fabric `getHistoryForKey` method
- `getByRange` – this calls the Hyperledger Fabric `getStateByRange` method
- `getByRangeWithPagination` – this calls the Hyperledger Fabric `getStateByRangeWithPagination` method

For more information, see: SDK Methods.

### SDK Methods

> **Note:**
>
> Beginning with version 21.3.2, the way to access the ORM methods has changed. Run the `ochain --version` command to determine the version of Blockchain App Builder.

In previous releases, the ORM methods were inherited from the `OchainModel` class. In version 21.3.2 and later, the methods are defined on the `Model` class of Context (`Ctx`) object. To call these methods, access them by using `this.Ctx.Model.<method_name>`.

The following example shows a method call in previous releases:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier){
    return await asset.save();
}
```

The following example shows a method call from the version 21.3.2 and later:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
      return await this.Ctx.Model.save(asset);
}
```

After you upgrade to version 21.3.2, make this change in all chaincode projects that you created with an earlier version of Blockchain App Builder. If you use the `sync` command to synchronize changes between the specification file and your source code, the changes are automatically brought to your controller for the ready-to-use methods. You still need to manually resolve any conflicts.

**save**

The `save` method adds the caller `asset` details to the ledger.

This method calls the Hyperledger Fabric `putState` internally. All marshalling/ unmarshalling is handled internally. The `save` method is part of the `Model` class, which you access by using the `Ctx` object.

```
Ctx.Model.save(asset: <Instance of Asset Class> , extraMetadata?:
any) : Promise <any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}
```

**get**

The `get` method is a method of `OchainModel` class which is inherited by the concrete model classes of `{chaincodeName}.model.ts`. The `get` method is part of the `Model` class, which you access by using the `Ctx` object.

If you would like to return any asset by the given `id`, use the generic controller method `getAssetById`.

```
Ctx.Model.get(id: string, modelName: <Model Asset Class Name>) :
Promise<asset>
```

Parameters:

- `id : string` – Key used to save data into the ledger.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise: <Asset>` - If the `modelName` parameter is not provided and data exists in ledger, then `Promise<object>` is returned. If the `id` parameter does not exist in ledger, an error message is returned. If the `modelName` parameter is provided, then an object of type `<Asset>` is returned. Even though any asset with given `id` is returned from the ledger, this method will take care of casting into the caller `Asset` type. If the asset returned from the ledger is not of the `Asset` type, then it throws an error. This check is done by the read-only `assetType` property in the `Model` class.

Example:

```
@Validator(yup.string())
public async getSupplierById(id: string) {
    const asset = await this.Ctx.Model.get(id, Supplier);
    return asset;
}
```

In the example, `asset` is of the type `Supplier`.

**update**
The `update` method updates the caller `asset` details in the ledger. This method returns a promise.

This method calls the Hyperledger Fabric `putState` internally. All the marshalling/ unmarshalling is handled internally. The `update` method is part of the `Model` class, which you can access by using the `Ctx` object.

```
Ctx.Model.update(asset: <Instance of Asset Class> , extraMetadata?: any) :
Promise <any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await this.Ctx.Model.update(asset);
}
```

**delete**

This deletes the asset from the ledger given by `id` if it exists. This method calls the Hyperledger Fabric `deleteState` method internally. The `delete` method is part of the `Model` class, which you can access by using the `Ctx` object.

```
Ctx.Model.delete(id: string): Promise <any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any>` - Returns a promise on completion.

Example:

```
@Validator(yup.string())
public async deleteSupplier(id: string) {
    const result = await this.Ctx.Model.delete(id);
    return result;
}
```

**history**

The `history` method is part of the `Model` class, which you can access by using the `Ctx` object. This method returns the asset history given by `id` from the ledger, if it exists.

This method calls the Hyperledger Fabric `getHistoryForKey` method internally.

```
Ctx.Model.history(id: string): Promise <any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any[]>` - Returns any [] on completion.

Example

```
@Validator(yup.string())
public async getSupplierHistoryById(id: string) {
    const result = await this.Ctx.Model.history(id);
    return result;
}
```

Example of the returned asset history for `getSupplierHistoryById`:

```
[
    {
```

```
        "trxId":
"8ef4eae6389e9d592a475c47d7d9fe6253618ca3ae0bcf77b5de57be6d6c3829",
        "timeStamp": 1602568005,
        "isDelete": false,
        "value": {
            "assetType": "supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 10,
            "license": "abcdabcdabcd",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    },
    {
        "trxId":
"92c772ce41ab75aec2c05d17d7ca9238ce85c33795308296eabfd41ad34e1499",
        "timeStamp": 1602568147,
        "isDelete": false,
        "value": {
            "assetType": "supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 15,
            "license": "valid license",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    }
]
```


**getByRange**

The `getByRange` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRange` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ] >`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, use the generic controller method `getAssetsByRange`.

```
Ctx.Model.getByRange(startId: string, endId: string, modelName: <Asset Model
Class Name> ): Promise <any>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.

- `endId : string` – Ending key of the range. Excluded of the range.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

Example:

```
@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string) {
    const result = await this.Ctx.Model.getByRange(startId, endId,
Supplier);
    return result;
}
```

**getByRangeWithPagination**

The `getByRangeWithPagination` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRangeWithPagination` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ] >`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, filtered by page size and bookmarks, use the generic controller method `getAssetsByRange`.

```
public async getByRangeWithPagination<T extends
OchainModel<T>>(startId: string, endId: string, pageSize: number,
bookmark?: string, instance?: new (data: any, skipMandatoryCheck:
boolean, skipReadOnlyCheck: boolean) => T): Promise<T[]>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded from the range.
- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.
- `modelName: <Model Asset Class Name>` – (Optional) Model asset class to return.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

**getId**
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` – Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
@Validator(yup.string(), yup.string())

public async customGetterForSupplier(license: string, name: string){
    let object = {
      license : license,
      name: name
    }
    const id = await this.Ctx.Model.getID(object);
    return this.Ctx.Model.get(id);
}
```

For token SDK methods, see the topics under Tokenization Support Using Blockchain App Builder.

### Controller

Main controller class extends `OchainController`. There is only one main controller.

```
export class TSProjectController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable from outside, the rest of them are hidden.

### Automatically Generated Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await this.Ctx.Model.save(asset);
}

@Validator(yup.string())
public async getSupplierById(id: string) {
    const asset = await this.Ctx.Model.get(id, Supplier);
    return asset;
}
```

```
@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await this.Ctx.Model.update(asset);
}

@Validator(yup.string())
public async deleteSupplier(id: string) {
    const result = await this.Ctx.Model.delete(id);
    return result;
}

@Validator(yup.string())
public async getSupplierHistoryById(id: string) {
    const result = await this.Ctx.Model.history(id);
    return result;
}

@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string) {
    const result = await this.Ctx.Model.getByRange(startId, endId,
Supplier);
    return result;
}
```

### Controller Method Details

Apart from the above model CRUD and non-CRUD methods, Blockchain App Builder provides out-of-the box support for other Hyperledger Fabric methods from our controller. These methods are:

- `getAssetById`
- `getAssetsByRange`
- `getAssetHistoryById`
- `query`
- `queryWithPagination`
- `generateCompositeKey`
- `getByCompositeKey`
- `getTransactionId`
- `getTransactionTimestamp`
- `getChannelID`
- `getCreator`
- `getSignedProposal`
- `getArgs`
- `getStringArgs`
- `getMspID`
- `getNetworkStub`

> **Note:**
>
> These methods are available with the `this` context in any class that extends the `OChainController` class.

For example:

```
public async getModelById(id: string) {
    const asset = await this.getAssetById(id);
    return asset;
}
@Validator(yup.string(), yup.string())
public async getModelsByRange(startId: string, endId: string) {
    const asset = await this.getAssetsByRange(startId, endId);
    return asset;
}
public async getModelHistoryById(id: string) {
    const result = await this.getAssetHistoryById(id);
    return result;
}
```

**getAssetById**

The `getAssetById` method returns asset based on `id` provided. This is a generic method and be used to get asset of any type.

```
this.getAssetById(id: string): Promise<byte[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <byte [ ]>` - Returns promise on completion. You have to convert `byte[]` into an object.

**getAssetsByRange**

The `getAssetsByRange` method returns all assets present from `startId` (inclusive) to `endId` (exclusive) irrespective of asset types. This is a generic method and can be used to get assets of any type.

```
this.getAssetsByRange(startId: string, endId: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< shim.Iterators.StateQueryIterator>` - Returns an iterator on completion. You have to iterate over it.

**`getAssetHistoryById`**
The `getAssetHistoryById` method returns history iterator of an asset for `id` provided.

```
this.getAssetHistoryById(id: string):
Promise<shim.Iterators.HistoryQueryIterator>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise<shim.Iterators.HistoryQueryIterator>` - Returns a history query iterator. You have to iterate over it.

**`query`**
The `query` method will run a Rich SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
this.query(queryStr: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `queryStr : string` - Rich SQL/Couch DB query.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

**`queryWithPagination`**
This method runs a Rich SQL/Couch DB query over the ledger, filtered by page size and bookmarks. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
public async queryWithPagination(query: string, pageSize: number,
bookmark?: string)
```

Parameters:

- `query : string` - Rich SQL/Couch DB query.
- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

**`generateCompositeKey`**
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
this.generateCompositeKey(indexName: string, attributes:
string[]): string
```

Parameters:

- `indexName : string` - Object Type of the key used to save data into the ledger.
- `attributes: string[ ]` - Attributes based on which composite key will be formed.

Returns:

- `string` - Returns a composite key.

**`getByCompositeKey`**
This method returns the asset that matches the key and the column given in the attribute parameter while creating composite key. `indexOfId` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`. Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
this.getByCompositeKey(key: string, columns: string[],
indexOfId: number): Promise<any []>
```

Parameters:

- `key: string` – Key used to save data into ledger.
- `columns: string[ ]` - Attributes based on key is generated.
- `indexOfId: number` - Index of attribute to be retrieved from Key.

Returns:

- `Promise< any [ ]` - Returns any `[]` on completion.

**`getTransactionId`**
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
this.getTransactionId(): string
```

Parameters:

- none

Returns:

- `string` - Returns the transaction ID for the current chaincode invocation request.

**`getTransactionTimestamp`**
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
this.getTransactionTimestamp(): Timestamp
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Timestamp` - Returns the timestamp when the transaction was created.

**`getChannelID`**
Returns the channel ID for the proposal for chaincode to process.

```
this.getChannelID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the channel ID.

**`getCreator`**
Returns the identity object of the chaincode invocation's submitter.

```
this.getCreator(): shim.SerializedIdentity
```

Parameters:

- none

Returns:

- `shim.SerializedIdentity` - Returns identity object.

**`getSignedProposal`**
Returns a fully decoded object of the signed transaction proposal.

```
this.getSignedProposal():
shim.ChaincodeProposal.SignedProposal
```

Parameters:

- none

Returns:

- `shim.ChaincodeProposal.SignedProposal` - Returns decoded object of the signed transaction proposal.

**`getArgs`**
Returns the arguments as array of strings from the chaincode invocation request.

```
this.getArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

**`getStringArgs`**
Returns the arguments as array of strings from the chaincode invocation request.

```
this.getStringArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

**`getMspID`**
Returns the MSP ID of the invoking identity.

```
this.getMspID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

**`getNetworkStub`**
The user can get access to the shim stub by calling `getNetworkStub` method. This will help user to write its own implementation of working directly with the assets.

```
this.getNetworkStub(): shim.ChaincodeStub
```

Parameters:

- none

Returns:

- `shim.ChaincodeStub` - Returns chaincode network stub.

**`invokeCrossChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
this.invokeCrossChaincode(chaincodeName: string, methodName: string,
args: string[], channelName: string): Promise<any>
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `Promise<any>` - Returns a JSON object that contains three fields:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

**`invokeChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
this.invokeChaincode(chaincodeName: string, methodName: string, args:
string[], channelName: string): Promise<any>
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `Promise<any>` - Returns a JSON object that contains three fields:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

### Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

```
/**
*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string()}
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
@Validator(yup.string(), yup.number()}
public async fetchRawMaterial(supplierId: string, rawMaterialSupply: number)
{
}
@Validator(yup.string(), yup.string(), yup.number())
public async getRawMaterialFromSupplier(manufacturerId: string, supplierId:
string, rawMaterialSupply: number) {
}
@Validator(yup.string(), yup.number(), yup.number())
public async createProducts(manufacturerId: string, rawMaterialConsumed:
number, productsCreated: number) {
}
public async sendProductsToDistribution() {
}
```

### Init Method

A custom `init` method is provided in the controller with an empty definition. If you use Blockchain App Builder to deploy or upgrade, the `init` method is called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v1.4.7 platform, the `init` method is also called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v2.x platform, you must call the `init` method manually. You can use a third-party tool such as Postman to call the `init` method manually.

```
export class TestTsProjectController extends OchainController {
    public async init(params: any) {
        return;
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

# Scaffolded Go Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project. The project contains automatically generated classes and functions, CRUD methods, SDK methods, automatic validation of arguments, marshalling/un-marshalling and transparent persistence capability (ORM).

If the chaincode project is in the Go language, the scaffolded project contains three main files:

- `main.go`
- `<chaincodeName>.model.go`
- `<chaincodeName>.controller.go`

All the necessary libraries are installed and packaged.

The `<chaincodeName>.model.go` file in the `model` subdirectory contains multiple asset definitions and the `<chaincodeName>.controller.go` file in the `controller` subdirectory contains the asset's behavior and CRUD methods. The various Go struct tags and packages in `model.go` and `controller.go` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

The scaffolded project can be found in `$GOPATH/src/example.com/<chaincodeName>`

**Reference:**

- Model
- Validators
- ORM
- SDK Methods
- Composite Key Methods
- Stub Method
- Other Methods
- Utility Package
- Controller
- Automatically Generated Methods
- Custom Methods
- Init Method

### Model

#### Asset Type Property

By default every struct will have an additional property called `AssetType`. This property can be useful in fetching only assets of this type. Any changes to this property is

ignored during create and update of asset. The property value by default is `<modelName>`.

```
type Supplier struct {
AssetType string 'json:"AssetType" default:"TestGoProject.Supplier"'

SupplierId            string      'json:"SupplierId"
validate:"string,mandatory" id:"true'
RawMaterialAvailable  int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
License               string      'json:"License" validate:"string,min=10"'
ExpiryDate            date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
Active                bool      'json:"Active" validate:"bool"
default:"true"'
Metadata              interface{} 'json:"Metadata,omitempty"'
}
```

### Validators

#### Id

`id:"true"`

This validator identifies the property which uniquely defines the underlying asset. The asset is saved by the value in this key. This validator automatically applies when a new Go project is scaffolded.

In the below screenshot `"SupplierId"` is the key for the supplier asset and has a tag property `id:"true"` for the SupplierId property.

```
type Supplier struct {
    Supplierld              string        'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int           'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string        'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date     'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool          'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{}   'json:"Metadata,omitempty"'
}
```

#### Derived

`derived:"strategy,algorithm,format"`

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `strategy`: takes values of `concat` or `hash`. Requires an additional parameter `algorithm` if `hash` is selected. The default algorithm is `sha256`; `md5` is also supported.

- `format`: takes an array of specification strings and values to be used by the strategy.

```
type Supplier struct{
   AssetType string 'json:"AssetType" final:"chaincode1.Supplier"'
   SupplierId   string  'json:"SupplierId" validate:"string" id:"true"
mandatory:"true"
derived:"strategy=hash,algorith=sha256,format=IND%1%2,License,Name"'
   Name         string  'json:"Name" validate:"string,min=2,max=4"'
   License      string  'json:"License" validate:"string,min=2,max=4"'
}
```

### Mandatory

`validate:"mandatory"`

This marks the following property as mandatory and cannot be skipped while saving to the ledger. If skipped it throws an error. In the below example, `"SupplierId"` has a `validate:"mandatory"` tag.

```
Type Supplier struct {
    Supplierld              string      'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string      'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool        'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{} 'json:"Metadata,omitempty"'
}
```

### Default

`default:"<param>"`

This states that the following property can have a default value. The default value in the default tag is used when the property is skipped while saving to the ledger. In the below example property, `Active` has a default value of `true`, provided as tag `default:"true"`

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
```

```
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

**Validate types**
Basic Go types are validated for a property by defining a validate tag. These are the validate tags based on types:

- string: `validate: "string"`
- date: `validate: "date"`
- number: `validate: "int"`
- boolean: `validate: "bool"`

**Min validator**
`validate:"min=<param>"`

Using the min validator, minimum value can be set for a property of type number and string.

For type int: In the example, `RawMaterialAvailable` property has a minimum value of 0 and if a value less than 0 is applied to `RawMaterialAvailable` an error will be returned.

For type string: For the string type minimum validator will check the length of the string with the provided value. Therefore, in the below example the `License` property has to be minimum 10 characters long.

Example:

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

**Max validator**
`validate:"max=<param>"`

Using the max validator, the maximum value can be set for a property of type number and string.

For type int: Like the min validator, for type int, if a value provided for the `structfield` is greater than the value provided in the validator then an error will be returned.

For type string: Like the min validator, max validator will also check the length of the string with given value. In the example, the `Domain` property has a maximum value of 50, so if the

`Domain` property has a string length more than 50 characters, then an error message will be returned.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=1-5"'
    Domain             string        'json:"Domain"
validate:"url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

**Date validators**
**Before validator:**

```
validate:"before=<param>"
```

The before validator validates a property of type `date` to have a value less than the specified in parameter.

In this example, the `ExpiryDate` property should be before `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Supplierld           string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License              string       'json:"License"
validate:"string,min=10"'
    ExpiryDate           date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active               bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata             interface{}  'json:"Metadata,omitempty"'
}
```

**After validator:**

```
validate:"after=<param>"
```

The before validator validates a property of type `date` to have a value greater than the specified in parameter.

In this example, the `CompletionDate` property should be after `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Manufacturerld        string       'json:"Manufacturerld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,max=8"'
    ProductsAvailable     int          'json:"ProductsAvailable"
validate:"int"'
    CompletionDate        date.Date    'json:"CompletionDate"
validate:"date,after=2020-06-26"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

### URL validator

`validate:"url"`

The URL validator will validate a property for URL strings.

In this example, the `Domain` property has to be a valid URL.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold" validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=1-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

### Regexp validator

`validate:"regexp=<param>"`

Regexp validator will validate property for the input regular expression.

In this example, the `PhoneNumber` property will validate for a mobile number as per the regular expression.

```
type Customer struct {
Customerld      string       'json:"Customerld" validate:"string,mandatory"
id:"true"'
Name            string       'json:"Name" validate:"string,mandatory"'
ProductsBought  int          'json:"ProductsBought" validate:"int"'
```

```
OfferApplied    int          'json:"OfferApplied"
validate :"int,nax=0"'
PhoneNumber     string       'json:"PhoneNumber"
validate:"string,regexp=A\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]
{4})$"'
Received        bool         'json:"Received" validate:"bool"'
Metadata        interface{}  'json:"Metadata,omitempty"'
}
```

**Multiple validators**
Multiple validators can be applied a property.

In this example, the `Domain` property has validation for a string, URL, and min and max string length.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

**ORM**

Transparent Persistence Capability or simplified ORM is captured in the `Model` class of the Context (`Ctx`) object. If your model calls any of the following SDK methods, access them by using `t.Ctx.Model`.

SDK methods that implement ORM are the following methods:

- `Save` – this calls the Hyperledger Fabric `PutState` method
- `Get` – this calls the Hyperledger Fabric `GetState` method
- `Update` – this calls the Hyperledger Fabric `PutState` method
- `Delete` – this calls the Hyperledger Fabric `DeleteState` method
- `History` – this calls the Hyperledger Fabric `GetHistoryForKey` method
- `GetByRange` – this calls the Hyperledger Fabric `GetStateByRange` method
- `GetByRangeWithPagination` – this calls the Hyperledger Fabric `GetStateByRangeWithPagination` method

### SDK Methods

Go chaincodes implement Transparent Persistence Capability (ORM) with the model package.

> **Note:**
>
> Beginning with version 21.2.3, the way to access the ORM methods has changed. Run the `ochain --version` command to determine the version of Blockchain App Builder.

In previous releases, the ORM methods were exposed as static methods in the model package. The methods are now defined on the model receiver, which holds the transaction stub. To call these methods, you use the model receiver held by the transaction context in the controller. You call these methods as `t.Ctx.Model.<method_name>` instead of `model.<method_name>`.

The following example shows `Save` and `Get` method calls in previous releases:

```
func (t *Controller) CreateSupplier(asset Supplier) (interface{}, error) {
  return model.Save(&asset)
}

func (t *Controller) GetSupplierById(id string) (Supplier, error) {
  var asset Supplier
  _, err := model.Get(id, &asset)
  return asset, err
}
```

The following example shows `Save` and `Get` method calls from the version 21.2.3 and later:

```
func (t *Controller) CreateSupplier(asset Supplier) (interface{}, error) {
  return t.Ctx.Model.Save(&asset)
}

func (t *Controller) GetSupplierById(id string) (Supplier, error) {
  var asset Supplier
  _, err := t.Ctx.Model.Get(id, &asset)
  return asset, err
}
```

After you upgrade to version 21.2.3, make this change in all chaincode projects that you created with an earlier version of Blockchain App Builder. If you use the `sync` command to synchronize changes between the specification file and your source code, the changes are automatically brought to your controller for the ready-to-use methods. You still need to manually resolve any conflicts.

The following ORM methods are exposed via the model package:

**`Get`**
Queries the ledger for the stored asset based on the given ID.

```
func Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `Id` - The ID of the asset which is required from the ledger.
- `result (interface{})` - This is an empty asset object of a particular type, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.
- `asset (interface)` - Empty asset object, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.

Returns:

- `interface {}` - Interface contains the asset in the form of `map[string]interface{}`. Before operating on this map, it is required to assert the obtained interface with type `map[string]interface{}`. To convert this map into an asset object, you can use the utility API `util.ConvertMaptoStruct` (see: Utility Package).
- `error` - Contains an error if returned, or is nil.

**`Update`**
Updates the provided asset in the ledger with the new values.

```
func Update(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj (interface)` - The object that is required to be updated in the ledger is passed by reference into this API with the new values. The input asset is validated and verified according to the struct tags mentioned in the model specification and then stored into the ledger.

Returns:

- `interface{}` - The saved asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

**`Save`**
Saves the asset to the ledger after validating on all the struct tags.

```
func Save(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj/args[0] (interface{})` - The object that needs to be stored in the ledger is passed by reference in this utility method.

- `metadata/args[1] (interface{})` - This parameter is optional. It has been given in order to facilitate you if you're required to store any metadata into the ledger along with the asset at the runtime. This parameter can be skipped if no such requirement exists.

Returns:

- `interface {}` - The asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

**`Delete`**
Deletes the asset from the ledger.

```
func Delete(Id string) (interface{}, error)
```

Parameters:

- `id (string)` - The ID of the asset which is required to be deleted from the ledger.

Returns:

- `interface {}` - Contains the asset being deleted in the form of `map[string]interface{}`.

**`GetByRange`**
Returns the list of assets by range of IDs.

```
func GetByRange(startKey string, endKey string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey (string)` - Starting ID for the range of objects which are required.
- `endkey (string)` - End of the range of objects which are required.
- `asset interface` - (optional) Empty array of assets, which is passed by reference. This array will contain the result from this method. To be used if type-specific result is required.

Returns:

- `[]map[string]interface{}` - This array contains the list of assets obtained from the ledger. You can access the objects iterating over this array and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains an error if returned, or is nil.

**`GetByRangeWithPagination`**
The `GetByRangeWithPagination` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`, filtered by page size and bookmark. This method calls the Hyperledger Fabric `GetStateByRangeWithPagination` method internally.

If the `modelName` parameter is not provided, the method returns `Promise<Object [ ]>`. If the `modelName` parameter is provided, then the method handles casting into the caller `Model` type. In the following example, the result array is of the type `Supplier`. If the asset returned from the ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

To return all the assets between the range `startId` and `endId`, filtered by page size and bookmarks, use the generic controller method `getAssetsByRange`.

```
func (m *Model) GetByRangeWithPagination(startKey string, endKey
string, pageSize int32, bookmark string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey : string` – Starting key of the range. Included in the range.
- `endkey : string` – Ending key of the range. Excluded from the range.
- `pageSize : number` – The page size of the query.
- `Bookmark : string` – The bookmark of the query. Output starts from this bookmark.
- `asset interface` – (Optional) An empty array of assets, passed by reference. This array will contain the result from this method. Use this parameter to get type-specific results.

Returns:

- `[]map[string]interface{}` – An array that contains the list of assets retrieved from the ledger. You can access the objects by iterating over this array and asserting the objects as `map[string]interface{}` and using a utility for conversion to an asset object.
- `error` – Contains an error if an error is returned, otherwise nil.

**GetHistoryById**
Returns the history of the asset with the given ID.

```
func GetHistoryByID(Id string) ([]interface{}, error)
```

Parameters:

- `Id (string)` - ID of the asset for which the history is needed.

Returns:

- `[]interface{}` - This slice contains the history of the asset obtained from the ledger in form of slice of `map[string]interface{}`. You can access each history element by iterating over this slice and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains the error if observed.

**`Query`**
The query method will run a SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func Query(queryString string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Input the query string.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

**`QueryWithPagination`**
The query method will run a SQL/Couch DB query over the ledger, filtered by page size and bookmark. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func (m *Model) QueryWithPagination(queryString string, pageSize int32,
bookmark string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Rich SQL/Couch DB query.
- `pageSize : number` - The page size of the query.
- `bookmark : string` - The bookmark of the query. Output starts from this bookmark.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

**`InvokeCrossChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
func InvokeCrossChaincode(chaincodeName string, method string, args
[]string, channelName string) (interface{}, error)
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.

- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `interface{}` - Returns a `map[string]interface{}` object that contains three keys:
  - `isValid` - `true` if the call is valid.
  - `payload` - The output returned by the cross-chaincode call, as a JSON object.
  - `message` - The message returned by the cross-chaincode call, in UTF-8 format.

Return Value Example:

```
{
      "isValid": true,
      "message": "Successfully invoked method [CreateAccount] on sub-
chaincode [erc721_go_453]",
      "payload": {
                  "AccountId":
"oaccount~6b83b8ab931f99442897dd04cd7a2a55f808686f49052a40334afe3753fda
4c4",
                  "AssetType": "oaccount",
                  "BapAccountVersion": 0,
                  "NoOfNfts": 0,
                  "OrgId": "appdev",
                  "TokenType": "nonfungible",
                  "UserId": "user2"
      }
}
```

**`InvokeChaincode`**
You can use this method in a chaincode to call a function in another chaincode. Both chaincodes must be installed on the same peer.

```
func InvokeChaincode(chaincodeName string, method string, args
[]string, channelName string) (interface{}, error)
```

Parameters:

- `chaincodeName` – The name of the chaincode to call.
- `methodName` - The name of the method to call in the chaincode.
- `arg` - The argument of the calling method.
- `channelName` - The channel where the chaincode to call is located.

Returns:

- `interface{}` - Returns a `map[string]interface{}` object that contains three keys:

- `isValid` - `true` if the call is valid.
- `payload` - The output returned by the cross-chaincode call, in UTF-8 format.
- `message` - The message returned by the cross-chaincode call, in UTF-8 format.

Return Value Example:

```
{
    "isValid": true,
    "message": "Successfully invoked method [CreateAccount] on sub-chaincode
[erc721_go_453]",
    "payload":
"{\"AssetType\":\"oaccount\",\"AccountId\":\"oaccount~c6bd7f8dcc339bf7144ea2e
1cf953f8c1df2f28482b87ad7895ac29e7613a58f\",\"UserId\":\"user1\",\"OrgId\":\"
appdev\",\"TokenType\":\"nonfungible\",\"NoOfNfts\":0,\"BapAccountVersion\":0
}"
}
```

### Composite Key Methods

**`GenerateCompositeKey`**
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
func GenerateCompositeKey(indexName string, attributes []string)
(string, error)
```

Parameters:

- `indexName (string)` - Object type of the composite key.
- `attrbutes ([]string)` - Attributes of the asset based on which the composite key has to be formed.

Returns:

- `string` - This contains the composite key result.
- `error` - Contains the error if observed.

**`GetByCompositeKey`**
This method returns the asset that matches the key and the column given in the parameters. The `index` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
func GetByCompositeKey(key string, columns []string, index int)
(interface{}, error)
```

Parameters:

- `key (string)` - Object type provided while creating composite key.
- `column ([]string)` - This is the slice of attributes on which the ledger has to be queried using the composite key.
- `index(int)` - Index of the attribute.

Returns:

- `Interface{}` - Contains the list of assets which are result of this method.
- `error` - Contains any errors if present.

### Stub Method

**`GetNetworkStub`**
This method will return the Hyperledger Fabric `chaincodeStub`.

You can get access to the shim stub by calling the `GetNetworkStub` method. This will help you write your own implementation working directly with the assets.

```
func GetNetworkStub() shim.ChaincodeStubInterface
```

Parameters:

- none

Returns:

- `shim.ChaincodeStubInterface` - This is the Hyperledger Fabric chaincode stub.

### Other Methods

- `GetTransactionId()`
- `GetTransactionTimestamp()`
- `GetChannelID()`
- `GetCreator()`
- `GetSignedProposal()`
- `GetArgs()`
- `GetStringArgs()`
- `GetCreatorMspId()`
- `GetId`

**`GetTransactionId`**
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
func GetTransactionId() string
```

Parameters:

- none

Returns:

- `string` - This contains the required transaction ID.

**`GetTransactionTimestamp`**
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
func GetTransactionTimestamp() (*timestamp.Timestamp, error)
```

Parameters:

- none

Returns:

- `timestamp.Timestamp` - Contains the timestamp required.
- `error` - Contains any errors if present.

**`GetChannelID`**
Returns the channel ID for the proposal for the chaincode to process.

```
func GetChannelID() string
```

Parameters:

- none

Returns:

- `string` - Contains the required channel ID as a string.

**`GetCreator`**
Returns the identity object of the chaincode invocation's submitter

```
func GetCreator() ([]byte, error)
```

Parameters:

- none

Returns:

- `[]byte` - Contains the required identity object serialized.
- `error` - Contains any errors if present.

**GetSignedProposal**
Returns a fully decoded object of the signed transaction proposal.

```
func GetSignedProposal() (*peer.SignedProposal, error)
```

Parameters:

- none

Returns:

- `*peer.SignedProposal` - Contains the required signed proposal object.
- `error` - Contains any errors if present.

**GetArgs**
Returns the arguments as array of strings from the chaincode invocation request.

```
func GetArgs() [][]byte
```

Parameters:

- none

Returns:

- `[][]byte` - Contains the arguments passed.

**GetStringArgs**
Returns the arguments intended for the chaincode Init and Invoke as a string array.

```
func GetStringArgs() []string
```

Parameters:

- none

Returns:

- `[]string` - Contains the required arguments as a string array.

**GetCreatorMspId**
Returns the MSP ID of the invoking identity.

```
func GetCreatorMspId() string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

**GetId**
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` - Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
func (t *Controller) CustomGetterForSupplier(License string, Name string)
(interface{}, error){
   var asset Supplier
   asset.License = License
   asset.Name = Name
   id,err := t.Ctx.Model.GetId(&asset)

   if err !=nil {
      return nil, fmt.Errorf("error in getting ID %v", err.Error())
   }
   return t.GetSupplierById(id)
}
```

### Utility Package

The following methods in the utility package may be useful:

**Util.CreateModel**
Parses the provided JSON string and creates an asset object of the provided type.

```
func CreateModel(obj interface{}, inputString string) error
```

Parameters:

- `inputString (string)` - The input JSON string from which the object is to be created.
- `obj (interface{})` - The reference of the object that is to be created from the JSON string. This object will store the created model which is also validated as per validator tags.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

**util.ConvertMapToStruct**
Convert the provided map into object of provided type.

```
func ConvertMapToStruct(inputMap map[string](interface{}), resultStruct
interface{}) error
```

Parameters:

- `inputMap (map[string](interface{}))` - Map which needs to be converted into the asset object.
- `resultStruct (interface{})` - The reference of the required asset object which needs to be generated from the map. Contains the result asset object required.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

For token SDK methods, see the topics under Tokenization Support Using Blockchain App Builder.

### Controller

The `Controller.go` file implements the CRUD and custom methods for the assets.

You can create any number of classes, functions, or files, but only those methods that are defined on chaincode struct are invokable from outside, the rest of them are hidden.

### Automatically Generated Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
//
//Supplier
//
func (t *ChainCode) CreateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputString)
    if err != nil {
        return nil, err
    }
    return model.Save(&obj)
}

func (t *ChainCode) GetSupplierById(id string) (interface{}, error) {
    asset, err := model.Get(id)
    return asset, err
}

func (t *ChainCode) UpdateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputstring)
    if err != nil {
        return nil, err
    }
return model.Update(&obj)
}

func (t *ChainCode) DeleteSupplier(id string) (interface{}, error) {
    return model.Delete(id)
```

```
}

func (t *ChainCode) GetSupplierHistoryById(id string) (interface{}, error) {
    historyArray, err := model.GetHistoryByld(id)
    return historyArray, err
}

func (t *ChainCode) GetSupplierByRange(startkey string, endKey string)
(interface{}, error) {
    assetArray, err := model.GetByRange(startkey, endKey)
    return assetArray, err
}
```

### Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

You can implement the functionality according to the business logic. If you add custom methods, add them to the controller file. If you add custom methods to the library instead of the controller file, your changes will be lost when the library folder contents are updated during the synchronization or chaincode upgrade processes.

```
//
//Custom Methods
//
/*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE
network.
*/
func (t *ChainCode) ExecuteQuery(inputQuery string) (interface{}, error) {
    resultArray, err := model.Query(inputQuery)
    return resultArray, err
}

func (t *ChainCode) FetchRawMaterial(supplierId string, rawMaterialSupply
int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) GetRawMaterialFromSupplier(manufacturerId string,
supplierId string, rawMaterialSupply int) (interface{} error) {
    return nil, nil
}

Func (t *ChainCode) CreateProducts(manufacturerId string,
rawMaterialConsumed int, productsCreated int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) SendProductsToDistribution() (interface{}, error) {
```

```
    return nil, nil
}
```

For Go chaincodes, every custom method should return two values: *empty interface*, *error*. For example:

```
func (t *Controller) FetchRawMaterial(supplierId string,
rawMaterialSupply int) (interface{}, error) {
    return nil, nil
}
```

**Init Method**

A custom `Init` method is provided in the controller with an empty definition. If you use Blockchain App Builder to deploy or upgrade, the `Init` method is called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v1.4.7 platform, the `Init` method is also called automatically. If you deploy or upgrade from the Oracle Blockchain Platform console on the Hyperledger Fabric v2.x platform, you must call the `Init` method manually. You can use a third-party tool such as Postman to call the `Init` method manually.

```
type Controller struct {
}
func (t *Controller) Init(args string) (interface{}, error)
    { return nil, nil
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

# Deploy Your Chaincode Using Visual Studio Code

Once your chaincode project is created, you can deploy it locally to the automatically generated Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.

## Deploy the Chaincode to a Local Hyperledger Fabric Network

Once you have created your chaincode project, you can test it in a local Hyperledger Fabric basic network.

When you install the Blockchain App Builder extension for VS Code, it automatically creates a Hyperledger Fabric network with a single channel. This will be listed as `Local Environment` in the **Environments** pane. You can't delete or modify this environment; you can just deploy chaincodes to it and rebuild it if it stops working correctly.

Blockchain App Builder chaincode deployment starts the Hyperledger Fabric basic network, other services, and installs and deploys the chaincode for you.

1. In the **Chaincode Details** pane, select **Deploy**.
2. In the deployment wizard:

- Ensure the correct chaincode name is selected.
- Select your target environment - for local deployment choose **Local Environment**.
- Select the channel you want to deploy to. A channel named "`mychannel`" is created by default with the extension's installation, and can be used for testing.
- Optionally enter any initial parameters that may be required.
- For token projects, the first time that you deploy you must enter a list of token admins as a parameter. The list is an array of `{user_id, org_id}` information that specifies the token admins. For the local Hyperledger Fabric network, use the value `Org1MSP` for the `org_id` field. For NFT chaincodes, the keys for the `adminList` parameter are `userId` and `orgId` for TypeScript and `UserId` and `OrgId` for Go. After the first time you deploy, you can supply an empty array for the `adminList` parameter or you can use the `adminList` parameter to add token admins. Other deployers who are not the first-time deployer must supply an empty array for the `adminList` parameter. To do so, open the `Init` parameter list in the deployment pane and then click the minus sign (–) button next to the `adminList` parameter, which will select an empty array.

3. Click **Deploy**.

When the chaincode has finished deploying, the **Output** console will state that it has successfully installed and deployed it on the given channel.

### Troubleshooting

You may encounter the following issues when running your chaincode project on a local network.

#### Missing Go permissions

While installing Go chaincode project in local network, you might see an error similar to the following in the **Output** console:

```
INFO (Runtime): 2020/06/22 22:57:09 build started

INFO (Runtime): Building ....

INFO (Runtime): go build runtime/cgo: copying /Users/myname/Library/
Caches/go-build/f8/.….….d: open /usr/local/go/pkg/darwin_amd64/runtine/
cgo.a: permission denied

ERROR (Runtime): go build runtine/cgo: copying /Users/myname/Library/
Caches/go-build/f8/.….….d: open /usr/local/go/pkg/darwin_amd64/runtime/
cgo.a: permission denied

INFO (Runtime): An error occurred while building: exit status 1
```

This is due to missing permissions for Go. This error has been seen only in Mac OS. This is a known issue:

- https://github.com/golang/go/issues/37962
- https://github.com/golang/go/issues/24674
- https://github.com/udhos/update-golang/issues/15

Solution: change the permissions of your `$GOROOT` and try deploying again:

```
sudo chmod -R 777 /usr/local/go
```

**Deployment failure**
Due to deployment failure, corrupt deployment, a Docker peer container being full, or a Docker peer being killed in the local network, you may see an error similar to:

```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating
Chaincode GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container:
Failed to generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error
(404): manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting
container: Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API
error (404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown
exited: signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again.

Solution: Rebuild your runtime by selecting your local environment in the **Environments** pane, right-clicking and selecting **Rebuild Local Environment**. Attempt to deploy again.

**Environment Rebuild Required**
You may see an error similar to:

```
Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com:
error while creating mount source path '/host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-
cli/runtime/network/basic-network/config': mkdir /host_mnt/c/
Users/opc/.vscode/extensions/oracle.oracle-blockchain-1.4.0: operation
not permitted
Starting ca.example.com... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source
```

```
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/crypto-config/peerOrganizations/org1.example.com/ca': mkdir /
host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-blockchain-1.4.0:
operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source path
'/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-blockchain-1.4.0/
node_modules/@oracle/ochain-cli/runtime/network/basic-network/crypto-config/
peerOrganizations/org1.example.com/ca': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com ... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source path
'/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-blockchain-1.4.0/
node_modules/@oracle/ochain-cli/runtime/network/basic-network/crypto-config/
peerOrganizations/org1.example.com/ca': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while creating mount source path
'/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-blockchain-1.4.0/
node_modules/@oracle/ochain-cli/runtime/network/basic-network/crypto-config/
peerOrganizations/org1.example.com/ca': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Error in Chaincode deployment
```

You need to rebuild your local environment. In the App Builder **Environments** pane, right-click your local environment and select **Rebuild Local Environment**.

# Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network

After you've deployed and tested your chaincode project on a local network to ensure it's working as designed, you can deploy it to Oracle Blockchain Platform.

### Create a Connection Configuration to an Oracle Blockchain Platform Instance

You must have a Blockchain Platform instance up and running to complete this step.

1. In the Visual Studio Code **Environments** pane, click the **Create Environment** icon.
2. On the **Environments Details** wizard:
   - Enter the name for your remote environment.
   - Enter a description.
   - In **Remote Url**, enter the URL of the remote Oracle Blockchain Platform instance.
   - Enter the Oracle Identity Cloud Service user name and password for an Oracle Blockchain Platform user with the `ADMIN` or `REST_CLIENT` roles. To invoke the chaincode, only the `REST_CLIENT` role is necessary. To deploy or upgrade the chaincode, the IDCS user must also be assigned the `ADMIN` role. For more information about users and roles, see Set Up Users and Application Roles.
3. Click **Create** to save the environment. The user name and password are saved temporarily in the local Visual Studio Code session. If you close Visual Studio Code and then start a new Visual Studio Code session, you must enter the user name and password again.

### Deploy Your Chaincode

1. Select the chaincode project you want to deploy in the **Chaincodes** pane.
2. In the **Chaincode Details** pane, select **Deploy**.
3. In the deployment wizard, the name of the chaincode project should be pre-filled.
   - Select your target environment - for remote deployment choose the Oracle Blockchain Platform environment you set up previously.
   - Enter the name of the channel you want to deploy to.
   - Optionally set any required initial parameters.
4. Click **Deploy**.

After the chaincode is successfully deployed to the remote Oracle Blockchain Platform, the console log will show that the following events occurred:

- The Oracle Blockchain Platform details were successfully fetched.
- The list of peers was successfully fetched.
- The chaincode project was successfully installed.
- The chaincode project was successfully approved and committed.
- The chaincode was successfully deployed on each peer and the channel.

In an environment with multiple organizations, to re-deploy the chaincode on the same channel through a participant instance, use the console to deploy the chaincode.

#### Upgrading the Chaincode Project

Upgrading the chaincode is handled automatically by Blockchain App Builder. After you make changes to your chaincode, just deploy again, which will automatically upgrade the project for you. When you run the upgrade process, you can pass an empty array for the `adminList` parameter or use the `adminList` parameter to add token admins. If you are not the first-time deployer, you must supply an empty array for the `adminList` parameter. To pass an empty array for the `adminList` parameter, open the `Init` parameter list in the deployment pane and then click the minus sign (–) button next to the `adminList` parameter, which will select an empty array.

If your upgrade is successful, the log will show that the following events occurred:

- The Oracle Blockchain Platform details were successfully fetched.
- The list of peers was successfully fetched.
- A check was made to determine if the chaincode project is already installed, and if so, the version was fetched.
- The chaincode version was successfully upgraded (for example, from version 1.0 to 2.0).

In an environment with multiple organizations, to upgrade the chaincode, use the console and manually approve the chaincode from the participants.

## Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

You can package your chaincode projects for manual deployment to Oracle Blockchain Platform Cloud or Enterprise Edition.

The **Package** function creates an archive file that contains only the build and distribution files. The `binary`, `libs`, `node_modules`, and `test` folders from your chaincode project are not included. This archive file can be manually uploaded to Oracle Blockchain Platform for deployment.

1. Select your chaincode project in the **Chaincodes** pane.
2. Right-click and select **Package**.
3. Select a location to save the package to, and then click **Select Output Folder**.

Because of changes to software prerequisites, when you run the **Package** function for TypeScript chaincode, you are prompted for the provisioning date of the Oracle Blockchain Platform instance that you want to create the package for. The TypeScript chaincode created in Blockchain App Builder is not compatible with previous versions of Oracle Blockchain Platform without changes to the underlying infrastructure. Blockchain App Builder packages the chaincode infrastructure accordingly based on your selection.

When the command completes successfully, the location of the package is returned.

## Test Your Chaincode Using Visual Studio Code

If your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode

development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

## Test Your Chaincode on a Local Hyperledger Fabric Network

Once your chaincode project is running on a local network, you can test it.

Blockchain App Builder contains a built-in wizard to assist you with invoking or querying your chaincode.

1. Select your chaincode project in the **Chaincodes** pane. In the **Chaincode Details** pane, select **Execute**. The chaincode name should already be selected. Ensure the target environment is set to **Local Environment** and the channel will default to the only channel available.
2. In the **Function** field, select your method from the drop-down list. Every method available in the chaincode is listed.
3. In the **Function Param** field, select the **More Actions (...)** button. This will launch a window with available properties for your selected method. Enter the properties, click **Omit** for any non-mandatory property you don't want submitted when you invoke your method, and click **Save**.
4. Click **Invoke**.

The **Output** console window will show that the function has been invoked. Alternatively, in the **Chaincode Actions** pane, the **Function Output** window displays the output. Click the **More Actions (...)** button to see this output formatted.

If you want to save the method and parameters you just ran, you can click **Save** and enter a name and description for it. It will be saved in your chaincode project in the `Queries` folder. To use it again, right-click it and select **Open**.

If you make any changes to the controller file that would alter the methods, select the **Reload** icon at the top of the **Chaincode Execute** pane. The change should now be reflected in the **Function** drop-down list.

> **Note:**
>
> If you don't want to use the wizard for testing, you can also run the Blockchain App Builder command line tools in the Visual Studio Code **Terminal** window. Follow the instructions provided here to test with the command line: Test Your Chaincode on a Local Hyperledger Fabric Network.

### Testing Multiple Token Users Locally

To test a token project with multiple users locally, you can use the `tokenUser` property to change the caller of each transaction. Every scaffolded chaincode project includes a `.ochain.json` file, which stores metadata of the chaincode. You change the caller by updating the value of `tokenUser` field in the `.ochain.json` file.

```
{
  "name": "digiCurrCC",
  "description": "Chaincode package for digiCurrCC",
  "chaincodeName": "digiCurrCC",
  "chaincodeType": "node",
```

```
  "configFileLocation": "/Users/user1/token.yml",
  "appBuilderVersion": "21.2.3",
  "nodeVersion": "v12.18.1",
  "tokenUser": "admin"
}
```

When a project is scaffolded, the `tokenUser` property is set to the default `admin` user of the local network. To change the caller of a transaction, change the `tokenUser` property to match the `user_id` property that was set when the account was created when the `createAccount` (TypeScript) or `CreateAccount` (Go) method was called.

**Automatic Installation and Deployment After Update**

Whenever you update and save your chaincode, the changes will be compiled, installed and deployed automatically. There is no need to strip down or bring up the local network again. All projects will be automatically compiled and deployed on every change.

## Testing Lifecycle Operations on a Remote Oracle Blockchain Platform Network

Once your chaincode project has successfully deployed to your remote Oracle Blockchain Platform network, you can test it as described in Test Your Chaincode on a Local Hyperledger Fabric Network.

You can use the same invoke and query commands to perform all method transactions on a remote Oracle Blockchain Platform Cloud or Enterprise Edition network; everything supported on the local network is also supported on the remote network. Select the Oracle Blockchain Platform instance as your target environment when executing your tests.

**Testing Token Projects on a Remote Oracle Blockchain Platform Network**

You can test chaincode projects that work with tokens by using Blockchain App Builder, the Oracle Blockchain Platform REST proxy, or the Hyperledger Fabric SDK.

**Blockchain App Builder**

You can use the Visual Studio Code extension to invoke transactions with multiple user to test token chaincodes. To test with multiple users, change the authorization parameters (user name and password) in the Environments tab and then save the environment. While invoking transactions, select the same environment from the drop-down list and then execute the transaction.

**Oracle Blockchain Platform REST Proxy**

You can use the REST proxy in Oracle Blockchain Platform to run your a token chaincode on a remote Oracle Blockchain Platform network. Use any REST Proxy client, such as Postman REST Client, to test your chaincode project.

To test multiple users, change the authorization parameters (user name and password) in your REST client, or connect to a different instance of Oracle Blockchain Platform.

## Execute Berkeley DB SQL Rich Queries

If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

If you have used `executeQuery` in the `customMethods` section of the specification file, a corresponding `executeQuery` method will be created in the controller.

Specification file:

```
customMethods:
    - executeQuery
    - "fetchRawMaterial(supplierid: string, rawMaterialSupply: number)"
    - "getRawMaterialFromSupplier(manufacturerId: string, supplierld:
string, rawMaterialSupply: number)"
    - "createProducts(manufacturerId: string, rawMaterialConsumed:
number, productsCreated: number)"
    - "sendProductsToDistribution()"
```

Controller file:

```
**
*
* BDB sql rich queries can be executed in OBP CS/EE.
* This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string())
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
```

You can invoke this method to execute Berkeley DB SQL rich queries on Oracle Blockchain Platform network, ensuring that you select the Oracle Blockchain Platform environment that you created as your target environment when running the queries.

Example:

1. In the **Chaincode Details** pane, select **Execute**. The chaincode name, target environment, and channel should already be pre-filled from the deployment step.
2. In the **Function Name** field, select `executeQuery` from the drop-down list.
3. In the **Function Param** field, select the **More Actions (...)** button. This will launch a window where you can enter the query string. Enter the arguments for your query, and click **Save**.
4. Click **Query**.

The **Output** window and the will show the query being executed and the result.

```
ochain query executeQuery "SELECT key, valueJson FROM <STATE> WHERE
json_extract(valueJson, '$.rawMaterialAvailable') = 4"
```

The entire SQL query is taken in the argument, so you can make changes to your query on the fly.

## Generate CLI Commands from Queries

If you have saved queries in a chaincode project in Visual Studio Code, you can automatically generate the equivalent CLI commands.

You must have at least one saved query in a chaincode project to generate CLI commands for Mac OSX and Linux and for Microsoft Windows.

1. Expand the project in the **Chaincodes** pane.
2. Right-click **Queries**.
3. Click **Generate CLI Commands**.

Two text files are generated and displayed in the **Queries** section of the **Chaincodes** pane: `CLIcommandsForLinux.txt` and `CLIcommandsForWindows.txt`. Select the file name to open the file and show the corresponding CLI commands.

# Upgrading Chaincode Projects in Visual Studio Code

You can upgrade existing chaincode projects in Visual Studio Code to use the new features of the updated Blockchain App Builder.

For Go projects, upgrade to Go v1.20.10 before you run the command to upgrade your chaincode project.

To upgrade a chaincode project, right-click the project in the Oracle Blockchain Platform pane of Visual Studio Code, and then click **Upgrade**.

When you open the detail view of a chaincode project from a previous version of Blockchain App Builder, you are prompted with the following message: `New chaincode library is available. Would you like to upgrade?` You can select from three options:

- **Yes**. The chaincode project is upgraded. The chaincode files in the `lib` folder are replaced. If you made any changes to these library files, back up the modified files or track the changes that you made before you run the upgrade.
- **Later**. Upgrade notification is postponed for 24 hours. You are notified again after 24 hours.
- **No**. You are not prompted again to upgrade. You can still upgrade the project at any time by right-clicking and selecting **Upgrade**, as described previously.

After you upgrade a chaincode project, synchronize the specification file with the generated source code. For more information, see Synchronize Specification File Changes With Generated Source Code.

# Synchronize Specification File Changes With Generated Source Code

You can use the synchronization function to bring new changes from the specification file to the chaincode source files (model and controller). The function works with both TypeScript and Go projects.

Note:

- Synchronization is unidirectional: you can bring changes from your specification file into your chaincode project, but not the other way around. Changes made in your chaincode project remain as-is after the synchronizing process.

- The command works only if the chaincode project was scaffolded by using a specification file. Do not delete, rename or move the specification file if you plan to synchronize any changes from the specification file to the source code in future.
- During synchronization, the chaincode files in the `lib` folder are automatically upgraded. If you make any changes to these library files, back up the modified files or track the changes that you make before you use the synchronization function, so that you can apply those changes again after synchronization.

To synchronize your specification and chaincode files:

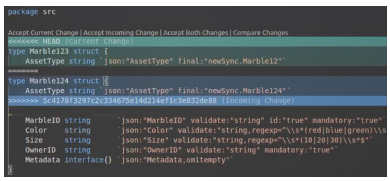
1. In the **Specifications** pane, select the specification file that you updated to open its **Specification Details** pane. At the top of the pane, click **Chaincodes** to open the pane showing which chaincodes were generated from the specification file.
2. Select the **Sync** check box beside each chaincode that you want to update with the new changes. You can synchronize more than one chaincode that was generated from a specification file at a time. Click **Synchronize**.

The chaincode projects now contain updated files.

**Resolving Conflicts**

Because you can edit both the synchronization files and chaincode files, it's possible to end up with conflicts where the updated specification file could overwrite a change that you've made to the chaincode file. In these cases, when you attempt to synchronize an error is displayed stating that there's a conflict. You can use the **Conflicts** pane to resolve these errors.

1. On the **Conflicts** pane, click the name of the chaincode file where the conflicts exist. The file opens in an editor with the conflicts highlighted.




   In the example shown, `Marble124` is in the specification file, and `Marble123` is in the chaincode model file.
2. Above the conflict is a list of options. Click **Accept Current Change** to override the specification file and use what is currently in the chaincode file. Click **Accept Incoming Change** to override the chaincode file and use what is currently in the specification file.
3. Return to the **Conflicts** pane. Select the **Sync** check box next to the conflict name, and then click **Confirm Changes**. If you have multiple conflicts, resolve all of them before before clicking **Confirm Changes**.

# Debugging from Visual Studio Code

Blockchain App Builder includes line-by-line debug support from Visual Studio Code for both TypeScript and Go projects.

On Microsoft Windows, configure Visual Studio Code to use Command Prompt as the default terminal instead of PowerShell. In the terminal menu in Visual Studio Code, click **Select Default Profile**, and then select **Command Prompt**.

Before you can debug your Go chaincode project, you must install the required Go tools in Visual Studio Code. In Visual Studio Code, open the Command Palette and then run the `Go: Install/Update Tools` command. Install all of the Go extensions that are listed.

Visual Studio Code uses Delve to debug Go. When you debug a Go chaincode for the first time, you will be prompted to install Delve. Accept the Delve installation before continuing. Visual Studio Code includes a built-in debugger for TypeScript.

To run line-by-line debugging:

1. Open your chaincode project in Visual Studio Code Explorer. In the **Chaincodes** pane, right-click your chaincode and select **Open in Explorer**.
2. Attach breakpoints to your code wherever necessary.
3. Go to the **Run** menu and click **Start Debugging**. This attaches the debugger. It may take several seconds for the debugger to attach to the chaincode.
4. Call any command from the Terminal which would execute your code.
   If you've been using the Visual Studio Code interface to test your chaincode so far, you can follow the invocation syntax outlined in Test Your Chaincode on a Local Hyperledger Fabric Network.

   The debugger will stop at your breakpoints. You can then start the debugging.
5. Restart debugging to reflect new changes.

Because the chaincode is running in debug mode, the hot deployment of new changes does not happen automatically. You must manually restart the debugging process, using the debug controls in Visual Studio Code, in order to make the latest changes take effect.

**Troubleshooting**

On Windows 11, you might encounter an error similar to the following:

```
dlv: failed to install dlv(github.com/go-delve/delve/cmd/dlv@latest): Error:
Command failed:
C:\Program Files (x86)\Go\bin\go.exe get -x github.com/go-delve/delve/cmd/
dlv@latest
# get https://proxy.golang.org/github.com/go-delve/delve/cmd/dlv/@v/list
# get https://proxy.golang.org/github.com/@v/list
# get https://proxy.golang.org/github.com/go-delve/@v/list
# get https://proxy.golang.org/github.com/go-delve/delve/cmd/@v/list
# get https://proxy.golang.org/github.com/go-delve/delve/@v/list
# get https://proxy.golang.org/github.com/@v/list: 410 Gone (0.420s)
# get https://proxy.golang.org/github.com/go-delve/delve/cmd/@v/list: 410
Gone (1.040s)
# get https://proxy.golang.org/github.com/go-delve/@v/list: 410 Gone (1.062s)
# get https://proxy.golang.org/github.com/go-delve/delve/cmd/dlv/@v/list:
410 Gone (1.066s)
# get https://proxy.golang.org/github.com/go-delve/delve/@v/list: 200 OK
(1.448s)
go: found github.com/go-delve/delve/cmd/dlv in github.com/go-delve/delve
v1.8.3C:\Users\<UserName>\go\pkg\mod\github.com\go-
delve\delve@v1.8.3\service\debugger\debugger.go:28:2:found packages native
(proc.go) and
your_operating_system_and_architecture_combination_is_not_supported_by_delve(
support_sentinel.go) in C:\Users\Asus\go\pkg\mod\github.com\go-
delve\delve@v1.8.3\pkg\proc\native
```

There is no workaround for this error at this time.

# Generate a Postman Collection Using Visual Studio Code

You can create a Postman collection that includes example payloads for all of your chaincode controller APIs.

Postman is a tool that you can use to work with and test REST APIs. The generate command creates a Postman collection that is based on the chaincode that was automatically generated from a declarative specification file. The Postman collection contains the payloads for all of the methods that are specified in the chaincode controller file. You can change the variable values in the Postman collection file to make REST API calls.

The generated Postman collection includes default values for all APIs in the controller. To learn more about Postman, see https://www.postman.com/. After you generate a Postman collection, you can directly import it and use it by changing the default values in the payload and variables.

To generate a Postman collection for a chaincode project in Visual Studio Code, complete the following steps.

1. Select the chaincode project in the Chaincodes pane.
2. Right-click the chaincode name and then select **Generate Postman Collection**.
3. Select a location to save the Postman collection to, and then click **Select Output Folder**.

If the specified Postman collection already exists, you are prompted whether to overwrite it.

### Postman Collection Structure

The generated Postman collection includes two types of requests, invoke requests and query requests:

- Invoke requests include all write operations, which use the endpoint `/transactions`
- Query requests include all get operations, which use the endpoint `/chaincode-queries`

To differentiate between getter and non-getter methods in the controller APIs, a decorator is used in TypeScript chaincodes and a comment is used in Go chaincodes. If you define a getter method in the controller, you must use the GetMethod decorator for TypeScript or the GetMethod comment for Go, as shown in the following table.

| TypeScript | Go |
|---|---|
| Every getter method has a `GetMethod` decorator:<br><br>`@GetMethod()`<br>`@Validator()`<br>`public async getAllTokenAdmins() {`<br>`  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ADMIN.getAllAdmins", "TOKEN");`<br>`  return await this.Ctx.ERC1155Admin.getAllAdmins();`<br>`}` | Every getter method has a `GetMethod` comment block:<br><br>`/**`<br>` * GetMethod`<br>` */`<br>`func (t *Controller) GetAllTokenAdmins() (interface{}, error) {`<br>`  auth, err := t.Ctx.Auth.CheckAuthorization("Admin.GetAllAdmins", "TOKEN")`<br>`  if err != nil && !auth {`<br>`  return nil, fmt.Errorf("error in authorizing the caller  %s", err.Error())`<br>`  }`<br>`  return t.Ctx.Admin.GetAllTokenAdmins()`<br>`}` |

Generated Postman collections include variables with default values, as shown in the following table.

| Variable Name | Description | Default Value | Context |
|---|---|---|---|
| `bc-url` | The REST proxy URL of the Oracle Blockchain Platform instance where the chaincode is deployed | `https://test-xyz-abc.blockchain.ocp.oraclecloud.com:7443/restproxy` | all chaincodes |
| `bc-channel` | The channel where the chaincode is deployed | `default` | all chaincodes |
| `bc-admin-user` | The name of the admin user (a user with the admin role that can access all POST requests). By default, this user is the caller of all POST requests in the chaincode | `bc-admin-user value` | all chaincodes |
| `bc-admin-password` | The password for the admin user | `bc-admin-password value` | all chaincodes |
| `bc-timeout` | The timeout value in the body of every POST request to indicate the timeout interval | `6000` | all chaincodes |

| Variable Name | Description | Default Value | Context |
|---|---|---|---|
| bc-sync | The sync value in the body of every POST request to indicate whether the request is synchronous or asynchronous | true | all chaincodes |
| bc-chaincode-name | The chaincode name, which is used in every POST request | chaincode name | all chaincodes |
| bc-org-id | The default orgId parameter for all POST requests | bc-org-id value | token chaincodes only |
| bc-user-id | The default userId parameter for all POST requests | bc-user-id value | token chaincodes only |
| bc-token-id | The default tokenId parameter for all POST requests | bc-token-id value | token chaincodes only |

In every generated request, all of the parameters with default values are generated. Functions that have struct/class parameters will have a placeholder object in the request body, as shown in the following examples.

**API with a struct/class parameter**

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "CreateArtCollectionToken",
        "{\"TokenId\":\"{{bc-token-id}}\",\"TokenDesc\":\"TokenDesc
value\",\"TokenUri\":\"TokenUri value\",\"TokenMetadata\":
{\"Painting_name\":\"Painting_name
value\",\"Description\":\"Description value\",\"Image\":\"Image
value\",\"Painter_name\":\"Painter_name
value\"},\"Price\":999,\"On_sale_flag\":true}",
        "quantity value"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

**API without a struct/class parameter**

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "CreateAccount",
        "{{bc-org-id}}",
        "example_minter",
        "true",
```

```
        "true"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

The default value for most API parameters is *parameter_name* value, with some exceptions. The following examples show some of the exceptions.

- The filters parameter in `GetAccountTransactionHistoryWithFilters`:

  ```
  "{\"PageSize\":20,\"Bookmark\":\"\",\"StartTime\":\"2022-01-16T15:16:36+00
  :00\",\"EndTime\":\"2022-01-17T15:16:36+00:00\"}"
  ```

- The filters parameter in `GetSubTransactionsByIdWithFilters`:

  ```
  "{\"PageSize\":20,\"Bookmark\":\"\"}"
  ```

A struct or class has different default values, as shown in the following table:

| Data Type | Default Value |
|---|---|
| boolean/bool | true |
| int/number | 999 |
| date | 2022-01-16T15:16:36+00:00 |
| other | *parameter_name* value |

### ERC-1155 Token Projects

The ERC-1155 standard includes common methods for both fungible and non-fungible tokens. The generated Postman collection for an ERC-1155 project that uses both fungible and non-fungible tokens includes two different POST requests, one for each type of token, for these common methods. If an ERC-1155 project uses only fungible or non-fungible tokens but not both types, then the generated Postman collection includes only one POST request for these common methods. The following table illustrates the generated API for the `AddRole` method.

| | Fungible Tokens | Non-Fungible Tokens |
|---|---|---|
| Request Name | `AddRole -For Fungible` | `AddRole -For NonFungible` |

| | Fungible Tokens | Non-Fungible Tokens |
|---|---|---|
| Request Body | {<br>"chaincode": "{{bc-chaincode-name}}",<br>"args": [<br>"AddRole",<br>"{{bc-org-id}}",<br>"{{bc-user-id}}",<br>"role value (for example, minter / burner)",<br><br>"{\"TokenId\":\"{{bc-token-id}}\"}"<br>],<br>"timeout": {{bc-timeout}},<br>"sync": {{bc-sync}}<br>} | {<br>"chaincode": "{{bc-chaincode-name}}",<br>"args": [<br>"AddRole",<br>"{{bc-org-id}}",<br>"{{bc-user-id}}",<br>"role value (for example, minter / burner)",<br><br>"{\"TokenName\":\"TokenName value\"}"<br>],<br>"timeout": {{bc-timeout}},<br>"sync": {{bc-sync}}<br>} |

# Troubleshoot Blockchain App Builder Visual Studio Code Extension

The following can be used to troubleshoot system problems with Blockchain App Builder Visual Studio Code extension.

**Prerequisites issues**
Errors can occur if you modify or upgrade any of the prerequisite software that is required by Blockchain App Builder. You can use the **Installed Dependencies** function to check that your installation of Blockchain App Builder still meets the prerequisites. On the Blockchain App Builder welcome page in Visual Studio Code, click **Installed Dependencies**. A table is displayed that shows you the required version and the actual installed version of components that Blockchain App Builder uses. Required components are indicated by an asterisk (*).

If the prerequisites check fails with errors and warnings when you attempt to install Blockchain App Builder, you might see an error similar to the following example in the output pane in Visual Studio Code:

```
Error:
Aborting installation. Error:
Found 1 error(s) in pre-requisites check, failed with following errors:
1. Golang version mismatch. Expected 1.20.x, but found 1.18.5.

Found 3 warning(s) in pre-requisites check.
1. Docker is not installed. Please install Docker >= 18.09.0. To
deploy chaincodes in the local environment,
please install the recommended version of Docker.
```

```
2. Docker Compose is not installed. Please install Docker Compose >= 1.23.0.
To deploy chaincodes in the
local environment, please install the recommended version of Docker Compose.
3. Git is not installed. To sync chaincodes, please install the Git
according to the documentation.
```

**Deployment failure**
Due to deployment failure, corrupt deployment, a Docker peer container being full, or a Docker peer being killed in the local network, you may see an error similar to:

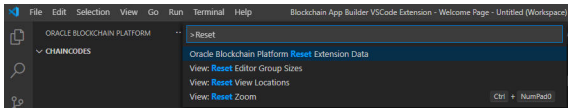
```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating Chaincode
GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container: Failed to
generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error (404):
manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting container:
Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API error
(404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown exited:
signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again.

Solution: Rebuild your runtime by selecting your local environment in the **Environments** pane, right-clicking and selecting **Rebuild Local Environment**. Attempt to deploy again.

**Resetting Extension Data**
It is possible for your Blockchain App Builder user data to get corrupted. This option clears your data from Blockchain App Builder without impacting anything stored locally.

1. Open the Command Palette from the **View** menu.
2. In the Command Palette, type `Reset Extension`.




3. Select **Oracle Blockchain Platform Reset Extension Data**. VS Code will clear the existing blockchain data and reload the default installation data. This will not affect the files stored locally in your system, but you will have to import them back into VS Code and reconfigure any environments you had previously set up.

**Mac OSX: Xcode**
After a Mac OSX upgrade, or if Xcode is not installed, you might see an error similar to the following in the error log:

```
gyp: No Xcode or CLT version detected!
gyp ERR! configure error
gyp ERR! stack Error: `gyp` failed with exit code: 1
gyp ERR! stack     at
```

- To work around this behavior, open a terminal window and run the following commands:

  ```
  sudo rm -rf $(xcode-select --print-path)
  xcode-select --install
  ```

# Tokenization Support Using Blockchain App Builder

You can use Blockchain App Builder to manage the complete life cycle of a token. You can tokenize existing assets and automatically generate token classes and methods to use for token lifecycle management.

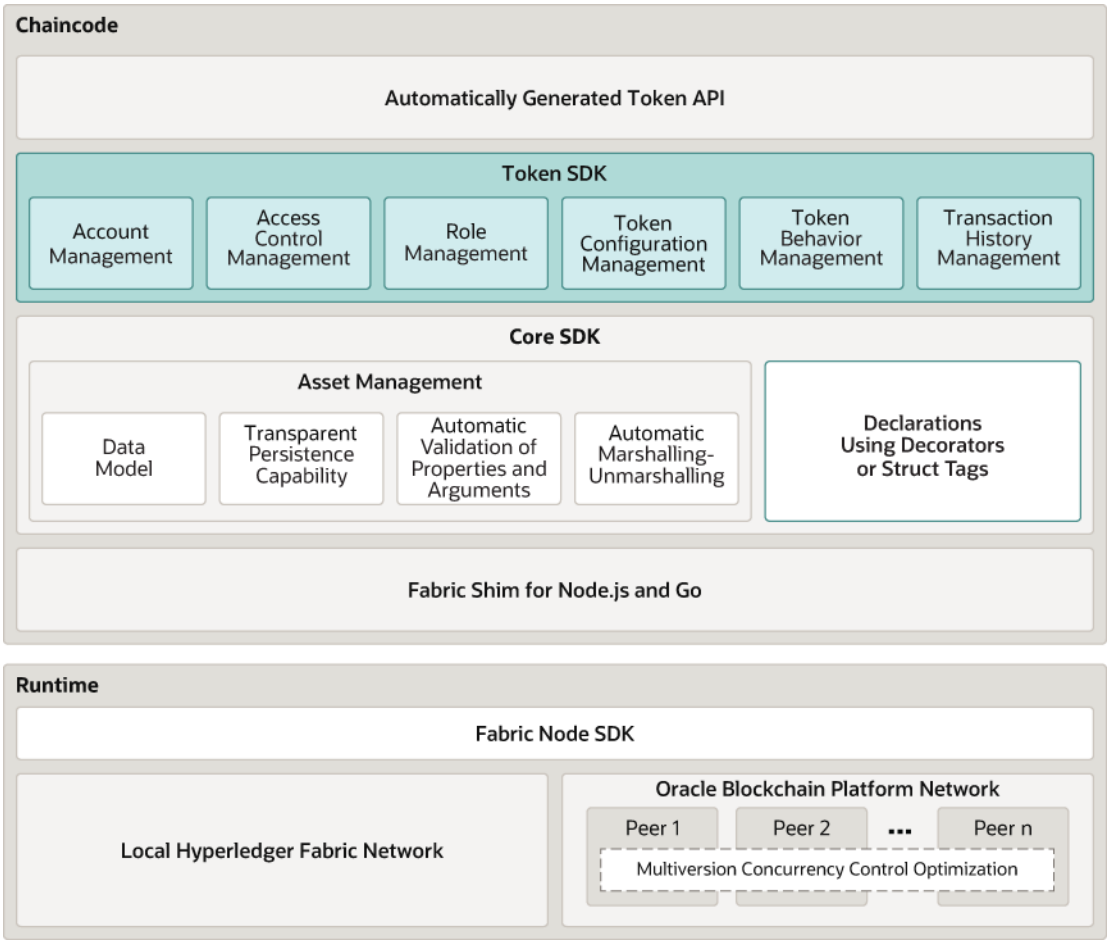
- Tokenization
- Tokens and the Account/Balance Model
- Token Standards
- Tokenization Flow
- Access Control
- MVCC Optimization

### Tokenization

Tokenization is a process where physical or digital assets are represented by tokens, which can be transferred, tracked, and stored on a blockchain. By representing assets as tokens, you can use the blockchain ledger to establish the state and ownership of an asset, and use standard blockchain platform functions to transfer ownership of an asset.

Blockchain App Builder includes tokenization support: token classes and methods are automatically generated, and additional token methods are provided so that developers can create complex business logic for tokens. The automatically generated project contains token lifecycle classes and functions, CRUD methods, and additional token SDK methods, and supports automatic validation of arguments, marshalling/unmarshalling, and transparent persistence capability. You can use these controller methods to initialize tokens, control access, set up accounts, manage roles, and manage the life cycle of tokens.

The following diagram shows the token architecture implemented by Blockchain App Builder, including the token API and token SDK.




**Automatically Generated Token API**
Blockchain App Builder automatically generates methods to support tokens and token life cycles. You can use these methods to initialize tokens, manage roles and accounts, and complete other token lifecycle tasks without any additional coding.

**Token SDK**
The Token SDK includes methods that help you develop complex business logic for token applications.

**Multiversion Concurrency Control (MVCC) Optimization**
The MVCC optimization for token chaincode can reduce errors for transfer, mint, burn, and hold operations.

### Tokens and the Account/Balance Model

Blockchain App Builder supports fungible and non-fungible tokens. Fungible tokens have an interchangeable value. Any quantity of fungible tokens has the same value as any other equal quantity of the same class of token. Non-fungible tokens are unique. Tokens can also be either whole or fractional. Fractional tokens can be subdivided into smaller parts, based on a specified number of decimal places.

Tokens can also be described by behaviors. Supported behaviors for fungible tokens include: `mintable`, `transferable`, `divisible`, `holdable`, `burnable`, and `roles` (`minter`, `burner`, and

holder). Supported behaviors for non-fungible tokens include: mintable, transferable, singleton, indivisible, burnable, and roles (minter and burner).

The tokenization feature uses an account/balance model to represent tokenized assets as balances in an account. Accounts are similar to typical banking accounts, where deposits and transfers and other state transitions affect the balance of an account. The balance of every account is tracked globally, to ensure that transaction amounts are valid. The on-hold balance (for fungible tokens) and transaction history are also tracked.

Any user who possesses tokens or completes token-related operations at any point must have an account on the network. Every account is identified by a unique ID (account_id). The account ID is created by combining a user name or email ID (user_id) of the instance owner or the user who is logged in to the instance with the membership service provider ID (org_id) of the user in the current network organization. Ready-to-use methods are provided for account creation. Because the account ID includes the organization ID, users can be supported across multiple organizations.

### Token Standards

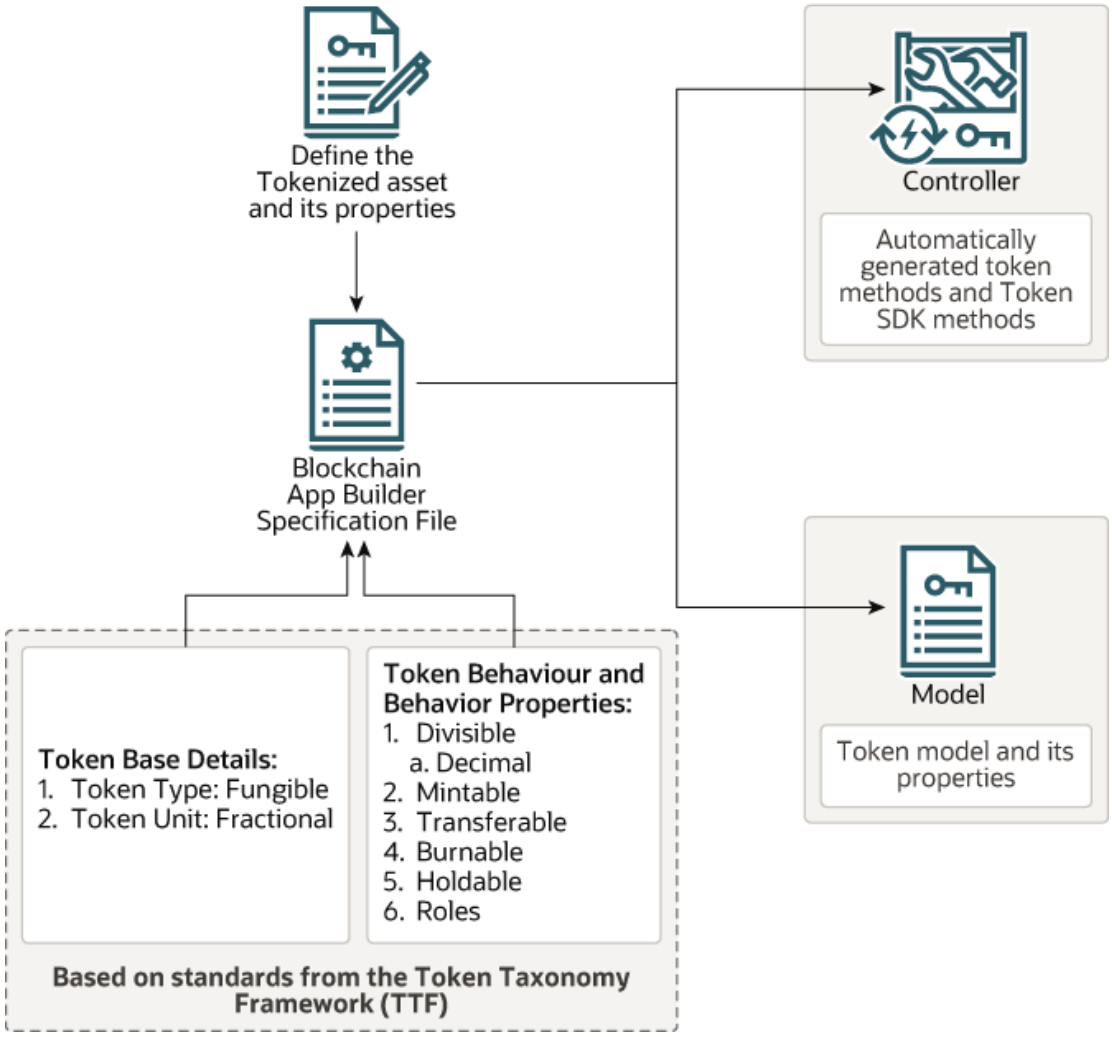
Blockchain App Builder extends the standards and classifications of the Token Taxonomy Framework, the ERC-721 standard, and the ERC-1155 standard to define the anatomy and behavior of tokens. ERC-1155 is a standard that supports both fungible and non-fungible tokens (NFTs). ERC-721 is a standard for NFTs. The Token Taxonomy Framework was developed by the Token Taxonomy Initiative. For more information, see Token Taxonomy Framework.

The following table describes the token types that Blockchain App Builder supports:

| Standard | Supported Token Types |
|---|---|
| Token Taxonomy Framework | • fractional fungible tokens |
| ERC-721 | • whole non-fungible tokens |
| ERC-1155 | • whole fungible tokens<br>• fractional fungible tokens<br>• whole non-fungible tokens<br>• fractional non-fungible tokens |

### Tokenization Flow

Because Blockchain App Builder supports tokenization by extending the input specification file syntax, you create token-specific projects the same way that you create other projects, either by using the CLI or in Visual Studio Code. For more information, see Input Specification File.




A typical tokenization flow follows these basic steps:

- Decide which token standard to use.
- Decide what token behaviors to specify (`mintable`, `transferable`, `divisible`, `indivisible`, `singleton`, `holdable`, `burnable`, and `roles`).
- Define the token asset and its properties in the input specification file.
- Scaffold the chaincode project from the input specification file. This creates a scaffolded project, including a model that contains the token asset definition and its properties and a controller that contains the token's behavior and methods.
- Deploy and test the chaincode project.

After you deploy a token-based project, the typical flow for creating tokens and completing lifecycle operations follows these steps:

- A token chaincode is deployed, and the users in the list passed to the initialization method become `Token Admin` users of the chaincode.
- A tokenized asset is initialized, which creates the `token_id`, a unique identifier for that particular instance of token.
- Accounts must be created for every user who will possess tokens or complete token-related operations.

- If the `roles` behavior is specified for the token, then roles must be added to users before they can complete token-related operations.
- Token life cycle methods can then be used, based on the behaviors that were specified for the token asset. For example, you can call a method to mint tokens for an account.

### Access Control

Tokenization support includes an access control feature that supports both role-based and ownership-based control mechanisms. With role-based control, users can call specific methods with an associated role such as `Token Admin` or `Token Minter`. With ownership-based control, you can restrict users from accessing assets that they do not own. With ownership-based access control, specific methods can be called by the users who own the assets, such as the `Token Owner` or `Account Owner`. For specific information on access control for methods, see the individual entries for the methods documented in the following topics:

- Scaffolded TypeScript Token Project for ERC-1155
- Scaffolded Go Token Project for ERC-1155
- Scaffolded TypeScript NFT Project for ERC-721
- Scaffolded Go NFT Project for ERC-721
- Scaffolded TypeScript Project for Token Taxonomy Framework
- Scaffolded Go Project for Token Taxonomy Framework

Role-based access control supports the following personas:

**Token Admin**
`Token Admin` users can be assigned when a token chaincode is deployed. The `Token Admin` user information is saved in the state database. A `Token Admin` user can grant and remove `Token Admin` privileges for other users. A `Token Admin` user cannot remove their own `Token Admin` privileges. A `Token Admin` user can assign the `Org Admin`, minter, burner, or notary role to any user.

**Org Admin**
The extended Token Taxonomy Framework methods support the `Org Admin` role. A `Token Admin` user can assign the `Org Admin` role to any user. `Org Admin` users have administrative privileges, but only within their organization. They can create accounts or see account balances, but only for users in their organization. `Org Admin` users who have a minter, burner, or notary role can assign that role to other users in their organization.

**Token Minter**
A user who is assigned the minter role is a `Token Minter`, and can mint tokens.

**Token Burner**
A user who is assigned the burner role is a `Token Burner`, and can burn tokens.

**Token Notary**
A user who is assigned the notary role is a `Token Notary`. A `Token Notary` acts as a third party in transactions between payers and payees. A `Token Notary` can either trigger the completion of a token transfer from a payer to a payee, or can instead return the tokens to the payer's account.

**Vault Manager**
A user who is assigned the vault role is the `Vault Manager`. The `Vault Manager` can unlock a locked NFT. The vault role is applicable only for the extended ERC-721 and ERC-1155 standards. Assigning the vault role to a user is a prerequisite for locking NFTs. Only one user per chaincode can be assigned the vault role.

Ownership-based access control supports the following personas:

**Account Owner**
Any user that has an account is an `Account Owner`.

**Token Owner**
Any user that currently owns a non-fungible token is the `Token Owner` of that token.

Role-based access control and ownership-based access control are also combined in some methods. For example, role-based access control lets a user with the minter role create tokens. With ownership-based access control, a non-fungible token owner can modify the custom properties of a token, but cannot modify the token metadata. When a user with the minter role creates a non-fungible token (NFT), they become the owner of the NFT. As the owner of that NFT, they can modify the custom properties (for an art collection token, the token price is a custom property). After the token creator transfers the NFT to another user, the second user becomes the owner, and the user who created the token is no longer the owner of the token. Because of ownership-based access control, the new owner can now update a custom property value, but the previous owner no longer can. Because of role-based access control, the previous owner can still mint an NFT, but the new user cannot.

You can also write your own access control functions, or disable access control. The automatically generated code that controls access is shown in the following examples.

TypeScript:

```
await this.Ctx.<Token Standard>Auth.checkAuthorization(...)
```

Go:

```
auth, err := t.Ctx.<Token Standard>Auth.CheckAuthorization(...)
```

> **Note:**
>
> To remove the automatically generated access control function, remove the previous line of code from your TypeScript or Go project.

**MVCC Optimization**

Hyperledger Fabric databases use multi-version concurrency control (MVCC) to avoid double-spending and data inconsistency. When the same state is updated, a new version of the record overwrites the old version. If there are concurrent requests to update the same key in a block, an `MVCC_READ_CONFLICT` error might be generated.

To reduce MVCC errors for transfer, mint, burn, and hold operations, you can enable the MVCC optimization for token chaincode. This optimization works on Oracle Blockchain Platform only. By default, the optimization is disabled. To enable the optimization, complete the applicable following step.

- CLI: Specify the Boolean `-m` or `--enable_mvcc_optimization` parameter with the `ochain init` command. By default, the parameter is set to `false`. To enable the optimization, add `-m true` to the `ochain init` command line.
- Visual Studio Code: When you create a chaincode, select **Enable MVCC optimization** on the Create Chaincode window.

To use the optimization with chaincode created in previous versions of Blockchain App Builder, complete the following steps:

1. After you install the latest version of Blockchain App Builder, upgrade the chaincode as described in Upgrading Chaincode Projects in the CLI and Upgrading Chaincode Projects in Visual Studio Code.
2. Edit the `.ochain.json` file in the root folder of the chaincode to set `enableMvccOptimization` to `true`.
3. Synchronize the chaincode, which adds the optimization and creates two new folders in the root folder of the the chaincode: `statedb` and `tokens`. For more information about synchronization, see Synchronize Specification File Changes With Generated Source Code and Synchronize Specification File Changes With Generated Source Code.

Other methods to work around `MVCC_READ_CONFLICT` errors including having the client application retry requests when this error is generated, or using a queue to capture concurrent requests before they are sent to the blockchain network. For more information, see Read-Write set semantics in the Hyperledger Fabric documentation.

> **Note:**
>
> The MVCC optimization does not work on hybrid networks that include both Oracle Blockchain Platform and Hyperledger Fabric peers, or for testing on a local Hyperledger Fabric network. Do not enable the MVCC optimization on hybrid networks, as this might lead to inconsistencies between peers.

# Token Taxonomy Framework

Blockchain App Builder supports an extended version of the Token Taxonomy Framework to work with fractional fungible tokens.

- Input Specification File for Token Taxonomy Framework
- Scaffolded TypeScript Project for Token Taxonomy Framework
- Scaffolded Go Project for Token Taxonomy Framework

## Input Specification File for Token Taxonomy Framework

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

You can define standard assets and token assets that are based on the Token Taxonomy Framework in the same specification file. You cannot define token assets based on more than one standard in the same specification file.

For information on including standard assets in the specification file, see Input Specification File.

The following sample specification files for fungible token assets are available in the Blockchain App Builder package:

- `FiatMoneyToken.yml`
- `LoyaltyToken-Go.yml`

In addition to the standard properties and sections, fungible token assets support the `behavior` and `anatomy` sections in the specification file. Fungible token assets also support the `standard` property. The following example shows the structure of a specification file for a fungible token asset based on the Token Taxonomy Framework:

```
assets:
    - name: OBPTOK # Asset name
      type: token  # Asset type

      anatomy:
          type: fungible # Token type
          unit: fractional # Token unit

      behavior: # Token behaviors
          - divisible:
                decimal: 2
          - mintable:
                max_mint_quantity: 1000
          - transferable
          - burnable
          - roles:
                minter_role_name: minter

      properties:
          - name: currency_name # Custom attribute to represent the token in
certain currency. This attribute is helpful for exchanging the tokens with
fiat currency.
            type: string

          - name: token_to_currency_ratio # Custom attribute to specify the
token to currency ratio. This attribute is helpful for exchanging the tokens
with fiat currency.
            type: number
```

**Table 7-5 Parameter Descriptions and Examples for a Fungible Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`type:`** | You must specify `type: token` in the `assets` section. | `assets:`<br>`- name: OBPTOK # Asset name`<br>`type: token # Asset type` |

**Table 7-5 (Cont.) Parameter Descriptions and Examples for a Fungible Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`standard:`** | The `standard` property represents the token standard to follow during chaincode generation. Only the `ttf+` value is supported for fungible tokens. If the `standard` property is not specified for a fungible token, the Token Taxonomy Framework (TTF) standard is followed. | `standard: ttf+ # Token standard` |
| **`anatomy:`** | The `anatomy` section has two mandatory parameters for fungible tokens:<br>• `type: fungible` A quantity of fungible tokens has the same value as another equal quantity of the same class of tokens.<br>• `unit: fractional` A fractional token can be subdivided into smaller units based on a specified number of decimal places. | `anatomy:`<br>`  type: fungible # Token type`<br>`  unit: fractional # Token unit` |

**Table 7-5 (Cont.) Parameter Descriptions and Examples for a Fungible Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`behavior:`** | This section describes the capabilities and restrictions of the token. The `mintable` and `transferable` behaviors are mandatory for fungible tokens.<br>• `mintable`: This mandatory behavior supports minting new token instances. The optional `max_mint_quantity` parameter specifies the total number of tokens that can be minted. If you do not specify the `max_mint_quantity` parameter, any number of tokens can be minted.<br>• `transferable`: This mandatory behavior supports transferring ownership of tokens.<br>• `divisible`: This optional behavior describes how tokens can be subdivided. The `decimal` parameter specifies the number of decimal places that can be used. The smallest fraction possible with the number of decimal places is the smallest unit of the token that | `behavior:`<br>`- mintable:`<br>`max_mint_quantity: 20000`<br>`- transferable`<br>`- divisible:`<br>`decimal: 1`<br>`- burnable`<br>`- holdable`<br>`- roles:`<br>`minter_role_name: minter`<br>`burner_role_name: burner`<br>`notary_role_name: notary` |

**Table 7-5 (Cont.) Parameter Descriptions and Examples for a Fungible Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | can be owned. If the `decimal` parameter is not specified, the default is zero decimal places.<br>• `burnable`: This optional behavior supports deactivating, or burning, tokens. Burning does not delete a token but instead places it in a permanent state where it cannot be used. Burning is not reversible.<br>• `holdable`: This optional behavior indicates whether token balances can be put on hold between a payer and payee.<br>• `roles`: This optional behavior restricts token behaviors to users with specific roles. Currently three roles are available: `minter_role_name`, `burner_role_name`, and `notary_role_name`. If you do not specify roles, then any user can act as a minter, burner, or notary. For example, if the burner role is not specified, any account user implicitly has the burner role. If the burner role is | |

**Table 7-5 (Cont.) Parameter Descriptions and Examples for a Fungible Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | specified, then during the token setup process, the `Token Admin` user must assign the burner role to other users explicitly. | |

To create multiple fungible token IDs that use different `max_mint_quantity` parameters, create a separate token asset for each token ID in the specification file, with a 1:1 relationship between token asset and token ID.

To create multiple fungible token IDs that use the same `max_mint_quantity` parameter or no `max_mint_quantity` parameter, create a single token asset in the specification file to use for all of the token IDs.

## Scaffolded TypeScript Project for Token Taxonomy Framework

Blockchain App Builder takes the input from your token specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates token lifecycle classes and functions, including CRUD and non-CRUD methods. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to tokens, see Scaffolded TypeScript Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated Token Methods
  - Custom Methods
- Token SDK Methods

### Model

Every tokenized model class extends the `Token` class, which in turn extends the `OchainModel` class. The `Token` class is imported from `../lib/token`. Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class.

```
import * as yup from 'yup';
import { Id, Mandatory, Validate, ReadOnly } from '../lib/decorators';
import { Token } from '../lib/token';

@Id('token_id')
export class Digicur extends Token<Digicur> {
```

```
    public readonly assetType = 'otoken';

    @Mandatory()
    @Validate(yup.string().required().matches(/^[A-Za-z0-9][A-Za-
z0-9_-]*$/).max(16))
    public token_id: string;

    @ReadOnly('digicur')
    public token_name: string;

    @Validate(yup.string().trim().max(256))
    public token_desc: string;

    @ReadOnly('fungible')
    public token_type: string;


@ReadOnly(["divisible","mintable","transferable","burnable","holdable",
"roles"])
    public behaviors: string[];

    @ReadOnly({minter_role_name: "minter", burner_role_name: "burner",
notary_role_name: "notary"})
    public roles: object;

    @ReadOnly({max_mint_quantity: 20000})
    public mintable: object;

    @ReadOnly({decimal: 1})
    public divisible: object;

    @Validate(yup.number())
    public token_to_currency_ratio: number;

    @Validate(yup.string())
    public currency_representation: string;

}
```

### Controller

The main controller class extends the `OchainController` class. There is only one main controller.

```
export class DigiCurrCCController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

### Automatically Generated Token Methods

Blockchain App Builder automatically generates methods to support tokens and token life cycles. You can use these methods to initialize tokens, manage roles and accounts, and complete other token lifecycle tasks without any additional coding. Controller methods must have a `@Validator(...params)` decorator to be invokable.

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Holdable Behavior
  - Burnable Behavior

### Methods for Access Control Management

**`addTokenAdmin`**
This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async addTokenAdmin(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization('ADMIN.addAdmin', 'TOKEN');
    return await this.Ctx.Admin.addAdmin(org_id, user_id);
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully added Admin (Org_Id: Org1MSP, User_Id: User1)"}
```

**removeTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async removeTokenAdmin(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization('ADMIN.removeAdmin',
'TOKEN');
    return await this.Ctx.Admin.removeAdmin(org_id, user_id);
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg": "Successfully removed Admin (Org_Id: Org1MSP, User_Id: User1)"}
```

**isTokenAdmin**

This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`. A `Token Admin` or `Org Admin` can call this function on any other user in the blockchain network. Other users can call this method only on their own accounts.

```
@Validator(yup.string(), yup.string())
  public async isTokenAdmin(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization("ADMIN.isUserTokenAdmin",
"TOKEN");
    return await this.Ctx.Auth.isUserTokenAdmin(org_id, user_id);
  }
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

**getAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by the `Token Admin` or any `Org Admin` of the chaincode.

```
@Validator()
public async getAllTokenAdmins() {
    await this.Ctx.Auth.checkAuthorization('ADMIN.getAllAdmins', 'TOKEN');
    return await this.Ctx.Admin.getAllAdmins();
}
```

Parameters:

- none

Returns:

- On success, an `admins` array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{"admins":[{"org_id":"Org1MSP","user_id":"admin"}]}
```

**addOrgAdmin**

This method adds a user as an `Org Admin` of the organization. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization.

```
@Validator(yup.string(), yup.string())
  public async addOrgAdmin(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization("ADMIN.addOrgAdmin", "TOKEN",
{ org_id });
    return await this.Ctx.Admin.addOrgAdmin(org_id, user_id);
  }
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully added Org Admin (Org_Id: Org1MSP, User_Id:
orgAdmin)"
}
```

**removeOrgAdmin**

This method removes a user as an `Org Admin` of the organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
@Validator(yup.string(), yup.string())
  public async removeOrgAdmin(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization("ADMIN.removeOrgAdmin",
"TOKEN", { org_id });
    return await this.Ctx.Admin.removeOrgAdmin(org_id, user_id);
  }
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as an `Org Admin` of the organization.

Return Value Example:

```
{
  "msg": "Successfully removed Org Admin (Org_Id Org1MSP User_Id
orgAdmin)"
}
```

**getOrgAdmins**

This method returns a list of all users who are an `Org Admin` of an organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of any organization.

```
  @Validator()
  public async getOrgAdmins() {
    await this.Ctx.Auth.checkAuthorization("ADMIN.getOrgAdmins",
"TOKEN");
    return await this.Ctx.Admin.getAllOrgAdmins();
  }
```

Parameters:

- none

Returns:

- On success, an array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{
    "admins": [
        {
            "org_id": "Org1MSP",
            "user_id": "orgadmin"
        },
        {
            "org_id": "Org1MSP",
            "user_id": "orgadmin1"
        },
        {
            "org_id": "Org1MSP",
            "user_id": "orgadmin2"
        }
    ]
}
```

### Methods for Token Configuration Management

**`init`**
This method is called when the chaincode is deployed or upgraded. Every `Token Admin` is identified by the `user_id` and `org_id` information in the mandatory `adminList` parameter. The `user_id` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `org_id` is the membership service provider (MSP) ID of the user in the current network organization.

Any `Token Admin` user can add and remove other `Token Admin` users by calling the `addAdmin` and `removeAdmin` methods.

```
public async init(adminList: TokenAdminAsset[]) {
    await this.Ctx.Admin.initAdmin(adminList);
    return;
}
```

Parameters:

- `adminList array` – An array of `{user_id, org_id}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

Parameter example, Mac OSX and Linux CLI:

```
'[{"user_id":"userid", "org_id":"OrgMSPId"}]'
```

Parameter example, Microsoft Windows CLI:

```
"[{\"user_id\":\"userid\", \"org_id\":\"OrgMSPId\"}]"
```

Parameter example, Oracle Blockchain Platform console:

```
["[{\"user_id\":\"userid\", \"org_id\":\"OrgMSPId\"}]"]
```

**`initialize<Token Name>Token`**
This method creates a token and initializes the token properties. The asset and its properties are saved in the state database. This method can be invoked only by a `Token Admin` of the chaincode.

```
@Validator(Digicur)
    public async initializeDigicurToken(token_asset: Digicur) {
        await this.Ctx.Auth.checkAuthorization('TOKEN.save', 'TOKEN');
        return await this.Ctx.Token.save(token_asset)
    }
```

Parameters:

- `asset: <Token Class>` – The token asset is passed as the parameter to this method. The properties of the token asset are described in the model file.

Returns:

- On success, a JSON representation of the token asset that was created.

Return Value Example:

```
{
    "assetType": "otoken",
    "token_id": "digiCurr101",
    "token_name": "digicur",
    "token_type": "fungible",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 1000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "DOLLAR",
    "token_to_currency_ratio": 1
}
```

**update<Token Name>Token**

This method updates token properties. After a token asset is created, only the `token_desc` property and custom properties can be updated. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(Digicur)
public async updateDigicurToken(token_asset: Digicur) {
    await this.Ctx.Auth.checkAuthorization('TOKEN.update', 'TOKEN');
    return await this.Ctx.Token.update(token_asset);
}
```

Parameters:

- `asset: <Token Class>` – The token asset is passed as the parameter to this method. The properties of the token asset are described in the model file.

Returns:

- On success, an updated JSON representation of the token asset.

Return Value Example:

```
{
    "assetType": "otoken",
    "token_id": "digiCurr101",
    "token_name": "digicur",
    "token_desc": "Digital Currency equiv of dollar",
    "token_type": "fungible",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 1000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "DOLLAR",
    "token_to_currency_ratio": 1
}
```

**getTokenDecimals**

This method returns the number of decimal places that were configured for a fractional token. If the `divisible` behavior was not specified for the token, then the default value is 0. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
@Validator(yup.string())
public async getTokenDecimals(token_id: string) {
    const token_asset = await this.getTokenObject(token_id);
    await this.Ctx.Auth.checkAuthorization('TOKEN.getDecimals',
'TOKEN');
    return {
        msg: `Token Id: ${token_id} has $
{this.Ctx.Token.getDecimals(token_asset)} decimal places.`
    };
}
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON string showing the number of token decimal places.

Return Value Example:

```
{"msg":"Token Id: digiCurr101 has 1 decimal places."}
```

**getTokenById**

This method returns a token object if it is present in the state database. This method can be called only by a `Token Admin` or an `Org Admin` of the chaincode.

```
@Validator(yup.string())
public async getTokenById(token_id: string) {
    await this.Ctx.Auth.checkAuthorization('TOKEN.get', 'TOKEN');
    const token = await this.getTokenObject(token_id);
    return token;
}
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON object that represents the token asset.

Return Value Example:

```
{
    "assetType": "otoken",
    "token_id": "digiCurr101",
```

```
    "token_name": "digicur",
    "token_desc": "Digital Currency equiv of dollar",
    "token_type": "fungible",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
        "burner_role_name": "burner",
        "notary_role_name": "notary"
    },
    "mintable": {
        "max_mint_quantity": 2000
    },
    "divisible": {
        "decimal": 1
    },
    "currency_name": "DOLLAR",
    "token_to_currency_ratio": 1
}
```

**getTokenHistory**

This method returns the token history for a specified token ID. Any user can call this method.

```
  @Validator(yup.string())
  public async getTokenHistory(tokenId: string) {
    await this.Ctx.Auth.checkAuthorization("TOKEN.getTokenHistory", "TOKEN");
    return await this.Ctx.Token.history(tokenId);
  }
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object that represents the token history.

Return Value Example:

```
[
    {
        "trxId":
"0d75f09446a60088afb948c6aca046e261fddcd43df416076201cdc5565f1a35",
        "timeStamp": "2023-09-01T16:48:41.000Z",
        "value": {
            "assetType": "otoken",
            "token_id": "token",
```

```
                "token_name": "fiatmoneytok",
                "token_desc": "updatedDesc",
                "token_standard": "ttf+",
                "token_type": "fungible",
                "token_unit": "fractional",
                "behaviors": [
                    "divisible",
                    "mintable",
                    "transferable",
                    "burnable",
                    "roles"
                ],
                "roles": {
                    "minter_role_name": "minter"
                },
                "mintable": {
                    "max_mint_quantity": 1000
                },
                "divisible": {
                    "decimal": 2
                }
            }
        },
        {
            "trxId":
"3666344878b043b65d5b821cc79c042ba52aec467618800df5cf14eac69f72fa",
            "timeStamp": "2023-08-31T20:24:55.000Z",
            "value": {
                "assetType": "otoken",
                "token_id": "token",
                "token_name": "fiatmoneytok",
                "token_standard": "ttf+",
                "token_type": "fungible",
                "token_unit": "fractional",
                "behaviors": [
                    "divisible",
                    "mintable",
                    "transferable",
                    "burnable",
                    "roles"
                ],
                "roles": {
                    "minter_role_name": "minter"
                },
                "mintable": {
                    "max_mint_quantity": 1000
                },
                "divisible": {
                    "decimal": 2
                }
            }
        }
]
```

**`getAllTokens`**
This method returns all tokens that are stored in the state database. This method can be called only by a `Token Admin` or an `Org Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@Validator()
public async getAllTokens() {
    await this.Ctx.Auth.checkAuthorization('TOKEN.getAllTokens', 'TOKEN');
    return await this.Ctx.Token.getAllTokens();
}
```

Parameters:

- none

Returns:

- On success, a JSON object that represents all token assets.

**`getTokensByName`**
This method returns all token objects with a specified name. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@Validator(yup.string())
public async getTokensByName(token_name: string) {
    await this.Ctx.Auth.checkAuthorization('TOKEN.getTokensByName', 'TOKEN');
    return await this.Ctx.Token.getTokensByName(token_name);
}
```

Parameters:

- `token_name: string` – The name of the tokens to retrieve. The name corresponds to the `token_name` property in the specification file. The value is the class name of the token.

Returns:

- On success, a JSON object of all token assets that match the name.

### Methods for Account Management

**`createAccount`**
This method creates an account for a specified user and token. An account must be created for any user who will have tokens at any point. Accounts track balances, on-hold balances, and transaction history. An account ID is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`org_id`) of the user in the current network organization. This method can

be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async createAccount(org_id: string, user_id: string,
token_type: string) {
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.createAccount",
"TOKEN", { org_id });
    return await this.Ctx.Account.createAccount(org_id, user_id,
token_type);
  }
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.
- `token_type: string` – The type of the token, which must be `fungible`.

Returns:

- On success, a JSON object of the account that was created. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
{
  "assetType": "oaccount",
  "account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",
  "bapAccountVersion": 0,
  "user_id": "admin",
  "org_id": "Org1MSP",
  "token_type": "fungible",
  "token_id": "",
  "token_name": "",
  "balance": 0,
  "onhold_balance": 0
}
```

**`associateTokenToAccount`**

This method associates a fungible token with an account. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the relevant organization.

```
  @Validator(yup.string(), yup.string())
  public async associateTokenToAccount(account_id: string, token_id:
string) {
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.associateToken",
"TOKEN", { account_id });
```

```
    return await this.Ctx.Account.associateToken(account_id, token_id);
  }
```

Parameters:

- `account_id: string` – The ID of the account.
- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON object of the updated account. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
{
    "assetType": "oaccount",
    "account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",
    "bapAccountVersion": 0,
    "user_id": "admin",
    "org_id": "Org1MSP",
    "token_type": "fungible",
    "token_id": "fungible",
    "token_name": "fiatmoneytok",
    "balance": 0,
    "onhold_balance": 0
}
```

**`getAccount`**

This method returns account details for a specified user and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
@Validator(yup.string(), yup.string(), yup.string())
public async getAccount(token_id: string, org_id: string, user_id: string) {
  const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
  await this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccount", "TOKEN",
{ account_id });
  return await this.Ctx.Account.getAccountWithStatus(account_id);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `account_id` – The ID of the user account.
- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current organization.
- `token_id` – The ID of the token.
- `token_name` – The name of the token.
- `balance` – The current balance of the account.
- `onhold_balance` – The current on-hold balance of the account.
- `bapAccountVersion` – An account object parameter for internal use.
- `status` – The current status of the user account.

Return Value Example:

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "account_id":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
  "user_id": "idcqa",
  "org_id": "appdev",
  "token_type": "fungible",
  "token_id": "t1",
  "token_name": "obptok",
  "balance": 0,
  "onhold_balance": 0
}
```

**`getAccountHistory`**

This method returns account history details for a specified user and token. This method can be called only by a `Token Admin` of the chaincode or the `AccountOwner` of the account.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getAccountHistory(token_id: string, org_id: string,
user_id: string) {
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.history", "TOKEN",
{ account_id });
    return await this.Ctx.Account.history(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, an array of JSON account objects that includes the following properties:
- `trxId` – The transaction ID of the transaction as returned by the ledger.
- `timeStamp` – The time of the transaction.
- `value` – A JSON string of the account object.

Return Value Example:

```
[
    {

"trxId":"2gsdh17fff222467e5667be042e33ce18e804b3e065cca15de306f837e416d7c3e",
      "timeStamp":1629718288,
      "value":{
         "assetType":"oaccount",

"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d
546d64f368642f622f",
         "user_id":"user1",
         "org_id":"Org1MSP",
         "token_id":"digiCurr101",
         "token_name":"digicur",
         "balance":100,
         "onhold_balance":0,
         "bapAccountVersion": 1
   },
   {

"trxId":"9fd07fff222467e5667be042e33ce18e804b3e065cca15de306f837e416d7c3e",
      "timeStamp":1629718288,
      "value":{
         "assetType":"oaccount",

"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d
546d64f368642f622f",
         "user_id":"user1",
         "org_id":"Org1MSP",
         "token_id":"digiCurr101",
         "token_name":"digicur",
         "balance":0,
         "onhold_balance":0,
         "bapAccountVersion": 0
      }
```

```
    }
]
```

**getAccountOnHoldBalance**

This method returns the current on-hold balance for a specified account and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getAccountOnHoldBalance(token_id: string, org_id:
string, user_id: string) {
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountOnHoldBalance",
"TOKEN", { account_id });
    return await this.Ctx.Account.getAccountOnHoldBalance(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the current on-hold balance.

Return Value Example:

```
{"msg":"Total Holding Balance is: 0","holding_balance":0}
```

**getAllAccounts**

This method returns a list of all accounts. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@Validator()
public async getAllAccounts() {
    await this.Ctx.Auth.checkAuthorization('ACCOUNT.getAllAccounts',
'TOKEN');
    return await this.Ctx.Account.getAllAccounts();
}
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

**getUserByAccountId**
This method returns user details (`org_id` and `user_id`) for a specified account. This method can be called by any user of the chaincode.

```
@Validator(yup.string())
public async getUserByAccountId(account_id: string) {
    return await this.Ctx.Account.getUserByAccountId(account_id);
}
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`org_id`, `token_id`, and `user_id`).

Return Value Example:

```
{
    "token_id": "digiCurr101",
    "user_id": "user1",
    "org_id": "Org1MSP"
}
```

**getAccountBalance**
This method returns the current balance for a specified account and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getAccountBalance(token_id: string, org_id: string, user_id:
string) {
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountBalance",
"TOKEN", { account_id });
    return await this.Ctx.Account.getAccountBalance(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the current account balance.

Return Value Example:

```
{"msg":"Current Balance is: 0","user_balance":0}
```

**`getAllOrgAccounts`**
This method returns a list of all token accounts that belong to a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
  @Validator(yup.string())
  public async getAllOrgAccounts(org_id: string) {
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getAllOrgAccounts", "TOKEN",
{ org_id });
    return await this.Ctx.Account.getAllOrgAccounts(org_id);
  }
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts for the specified organization.

Return Value Example:

```
[
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
        "valueJson": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
            "user_id": "idcqa",
            "org_id": "appdev",
            "token_type": "fungible",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "balance": 0,
            "onhold_balance": 0
        }
    },
    {
        "key":
"oaccount~620fcf5deb5fd5a65c0b5b10fda129de0f629ccd232c5891c130e24a574df
50a",
```

```
        "valueJson": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~620fcf5deb5fd5a65c0b5b10fda129de0f629ccd232c5891c130e24a574df50a",
            "user_id": "example_minter",
            "org_id": "appdev",
            "token_type": "fungible",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "balance": 0,
            "onhold_balance": 0
        }
    }
]
```

### Methods for Role Management

**`addRole`**

This method adds a role to a specified user and token. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization who also holds the specified role.

```
  @Validator(yup.string(), yup.string(), yup.string(), yup.string())
  public async addRole(token_id: string, role: string, org_id: string,
user_id: string) {
    const token_asset = await this.getTokenObject(token_id);
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("TOKEN.addRoleMember", "TOKEN",
{ token_id, org_id, role });
    return await this.Ctx.Token.addRoleMember(role, account_id, token_asset);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg":"Successfully added role 'minter' to Account Id:
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (Org-Id: Org1MSP, User-Id: user1)"}
```

**removeRole**

This method removes a role from a specified user and token. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization who also holds the specified role.

```
  @Validator(yup.string(), yup.string(), yup.string(), yup.string())
  public async removeRole(token_id: string, role: string, org_id:
string, user_id: string) {
    const token_asset = await this.getTokenObject(token_id);
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("TOKEN.removeRoleMember",
"TOKEN", { token_id, org_id, role });
    return await this.Ctx.Token.removeRoleMember(role, account_id,
token_asset);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg":"Successfully removed role 'minter' from Account Id:
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (Org-Id: Org1MSP, User-Id: user1)"}
```

**getAccountsByRole**

This method returns a list of all account IDs for a specified role and token. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async getAccountsByRole(token_id: string, role: string) {
```

```
    await this.Ctx.Auth.checkAuthorization('ROLE.getAccountsByRole', 'TOKEN');
    return await this.Ctx.Role.getAccountsByRole(token_id, role);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{"accounts":
["oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f36864
2f622f"]}
```

**`getAccountsByUser`**

This method returns a list of all account IDs for a specified organization ID and user ID. This method can be called only by the `Token Admin` of the chaincode, by the `Org Admin` of the specified organization, or by the `Account Owner` specified in the parameters.

```
  @Validator(yup.string(), yup.string())
  public async getAccountsByUser(org_id: string, user_id: string) {
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountsByUser",
"TOKEN", { org_id, user_id });
    return await this.Ctx.Account.getAccountsByUser(org_id, user_id);
  }
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{"accounts":
["oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f36864
2f622f"]}
```

**getUsersByRole**

This method returns a list of all users for a specified role and token. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async getUsersByRole(token_id: string, role: string) {
    await this.Ctx.Auth.checkAuthorization('ROLE.getUsersByRole',
'TOKEN');
    return await this.Ctx.Role.getUsersByRole(token_id, role);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of the user objects (`org_id`, `token_id`, and `user_id`).

Return Value Example:

```
{"users":
[{"token_id":"digiCurr101","user_id":"user1","org_id":"Org1MSP"}]}
```

**isInRole**

This method returns a Boolean value to indicate if a user and token has a specified role. This method can be called only by a `Token Admin` of the chaincode, the `AccountOwner` of the account, or an `Org Admin` of the specified organization.

```
  @Validator(yup.string(), yup.string(), yup.string(), yup.string())
  public async isInRole(token_id: string, org_id: string, user_id:
string, role: string) {
    const token_asset = await this.getTokenObject(token_id);
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("TOKEN.isInRole", "TOKEN",
{ account_id });
    return { result: await this.Ctx.Token.isInRole(role, account_id,
token_asset) };
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON string of the Boolean result.

Return Value Example:

```
{"result":"false"}
```

**getOrgAccountsByRole**
This method returns information about all accounts that have a specified role in a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
   @Validator(yup.string(), yup.string(), yup.string())
  public async getOrgAccountsByRole(token_id: string, role: string, org_id:
string) {
    await this.Ctx.Auth.checkAuthorization("ROLE.getOrgAccountsByRole",
"TOKEN", { org_id });
    return await this.Ctx.Role.getOrgAccountsByRole(token_id, role, org_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts with the specified role in the specified organization.

Return Value Example:

```
{
    "accounts": [

"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",

"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a850"
    ]
}
```

**getOrgUsersByRole**
This method returns information about all users that have a specified role in a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getOrgUsersByRole(token_id: string, role: string, org_id:
string) {
    await this.Ctx.Auth.checkAuthorization("ROLE.getOrgUsersByRole",
"TOKEN", { org_id });
```

```
    return await this.Ctx.Role.getOrgUsersByRole(token_id, role,
org_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all users with the specified role in the specified organization.

Return Value Example:

```
{
    "users": [
        {
            "token_id": "token",
            "user_id": "admin",
            "org_id": "Org1MSP"
        },
        {
            "token_id": "token",
            "user_id": "orgAdmin",
            "org_id": "Org1MSP"
        }
    ]
}
```

### Methods for Transaction History Management

**`getAccountTransactionHistory`**
This method returns an array of account transaction history details for a specified user and token. This method can be called only by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
 @Validator(yup.string(), yup.string(), yup.string())
  public async getAccountTransactionHistory(token_id: string, org_id:
string, user_id: string) {
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountTransactionHistory"
, "TOKEN", { account_id });
    return await
this.Ctx.Account.getAccountTransactionHistory(account_id, org_id,
user_id.toLowerCase());
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, an array of JSON account transaction objects that includes the following properties:
- `transaction_id` – The ID of the transaction.
- `transacted_account` – The account with which the transaction took place.
- `transaction_type` – The type of transaction.
- `transacted_amount` – The amount of the transaction.
- `timestamp` – The time of the transaction.
- `balance` – The account balance at the time of the transaction.
- `onhold_balance` – The on-hold balance at the time of the transaction.
- `token_id` – The ID of the token.
- `holding_id` – A unique identifier returned by the `holdTokens` method.

Return Value Example:


```
[
    {
        "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5",
        "transacted_amount": 20,
        "timestamp": "2021-08-17T06:04:24.000Z",
        "balance": 930,
        "onhold_balance": 0,
        "token_id": "digiCurr101",
        "transaction_type": "BULKTRANSFER",
        "sub_transactions": [
            {
                "transacted_account":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
                "transaction_type": "DEBIT",
                "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c4ca4238a0b923820dcc509a6f75849b",
                "transacted_amount": 10
            },
            {
                "transacted_account":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
```

```
c706e",
                "transaction_type": "DEBIT",
                "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775~c81e728d9d4c2f636f067f89cc14862c",
                "transacted_amount": 10
            }
        ]
    },
    {
        "transaction_id":
"otransaction~757864d5369bd0539d044caeb3bb4898db310fd7aa740f45a99387719
03d43da",
        "transacted_amount": 50,
        "timestamp": "2021-08-17T06:02:44.000Z",
        "balance": 950,
        "onhold_balance": 0,
        "token_id": "digiCurr101",
        "transacted_account":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
        "transaction_type": "DEBIT"
    }
]
```

**getAccountTransactionHistoryWithFilters**

This method returns an array of account transaction history details for a specified user and token. This method can be called only by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
  @Validator(yup.string(), yup.string(), yup.string(),
yup.object().nullable())
  public async getAccountTransactionHistoryWithFilters(token_id:
string, org_id: string, user_id: string, filters?: Filters) {
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountTransactionHistoryW
ithFilters", "TOKEN", { account_id });
    return await
this.Ctx.Account.getAccountTransactionHistoryWithFilters(account_id,
org_id, user_id.toLowerCase(), filters);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.

- `user_id: string` – The user name or email ID of the user.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetAccountTransactionHistoryWithFilters 'token1' 'appbuilder12'
'user_minter'
'{"PageSize":10,"Bookmark":"1","StartTime":"2022-01-25T17:41:42Z","EndTime":"20
22-01-25T17:59:10Z"}'
```


```
[
  {
    "transaction_id":
"otransaction~672897b5a4fa78b421c000e4d6d4f71f3d46529bfbb5b4be10bf5471dc35ce8
9",
    "transacted_amount": 5,
    "timestamp": "2022-04-20T15:46:04.000Z",
    "token_id": "tokenId",
    "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c274883b8",
    "transaction_type": "DEBIT",
    "balance": 90,
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~467bb67a33aaffca4487f33dcd46c9844efdb5421a2e7b6aa2d53152eb2c6d8
5",
    "transacted_amount": 5,
    "timestamp": "2022-04-20T15:45:47.000Z",
    "token_id": "tokenId",
    "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c248",
    "transaction_type": "DEBIT",
    "balance": 95,
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~c6d56ce54a9bbe24597d1d10448e39316dc6f16328bf3c5b0c8ef10e1dfeb39
7",
    "transacted_amount": 100,
    "timestamp": "2022-04-20T15:44:26.000Z",
    "token_id": "tokenId",
    "transacted_account":
"oaccount~deb5fb0906c40506f6c2d00c573b774e01a53dd91499e651d92ac4778b6add6a",
    "transaction_type": "MINT",
    "balance": 100,
    "onhold_balance": 0
```

```
  }
]
```

**getSubTransactionById**

This method returns an array of account transaction history details for a specified user and token. This method can be called only by the `Token Admin` of the chaincode or the `AccountOwner` of the account.

```
  @Validator(yup.string())
  public async getSubTransactionsById(transaction_id: string) {
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getSubTransactionsById",
"TOKEN", { transaction_id });
    return await
this.Ctx.Account.getSubTransactionsById(transaction_id);
  }
```

Parameters:

- `transaction_id: string` – The ID of the bulk transfer transaction.

Returns:

- An array of account subtransaction objects in JSON format for a specified bulk transfer transaction ID.

Example:

```
ochain invoke GetAccountSubTransactionHistory
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f8
64b9b'

[
    {
        "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c27488
3b8",
        "transaction_type": "DEBIT",
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d
7fe6cb8~c81e728d9d4c2f636f067f89cc14862c",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:52:21.000Z",
        "token_id": "token1",
        "balance": 80,
        "onhold_balance": 0
    },
    {
        "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c
248",
        "transaction_type": "DEBIT",
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d
```

```
7fe6cb8~c4ca4238a0b923820dcc509a6f75849b",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:52:21.000Z",
        "token_id": "token1",
        "balance": 85,
        "onhold_balance": 0
    }
]
```

**getSubTransactionsByIdWithFilters**

This method returns an array of account subtransaction history details for a specified transaction.

```
  @Validator(yup.string(), yup.object().nullable())
  public async getSubTransactionsByIdWithFilters(transaction_id: string,
filters?: SubTransactionFilters) {
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getSubTransactionsByIdWithFilters",
 "TOKEN", { transaction_id });
    return await
this.Ctx.Account.getSubTransactionsByIdWithFilters(transaction_id, filters);
  }
```

Parameters:

- `transaction_id: string` – The ID of the transaction.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Returns:

- An array of account subtransaction objects in JSON format for a specified bulk transfer transaction ID.

Example:

```
ochain invoke GetAccountSubTransactionHistoryWithFilters
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9b'
'{"PageSize":10,"Bookmark":"1"}'
```

```
[
  {
    "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d7fe6cb
8~c81e728d9d4c2f636f067f89cc14862c",
    "transacted_amount": 5,
    "timestamp": "2022-04-20T15:52:21.000Z",
    "token_id": "tokenId",
    "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c274883b8",
```

```
    "transaction_type": "DEBIT",
    "balance": 80,
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d
7fe6cb8~c4ca4238a0b923820dcc509a6f75849b",
    "transacted_amount": 5,
    "timestamp": "2022-04-20T15:52:21.000Z",
    "token_id": "tokenId",
    "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c
248",
    "transaction_type": "DEBIT",
    "balance": 85,
    "onhold_balance": 0
  }
]
```

**getTransactionById**

This method returns the history of a `Transaction` asset.

```
@Validator(yup.string())
    public async getTransactionById(transaction_id: string) {
        return await
this.Ctx.Transaction.getTransactionById(transaction_id);
    }
```

Parameters:

- `transaction_id string` – The ID of the transaction asset.

Returns:

- On success, an JSON array of the history for the transaction.

Return Value Example:

```
{
    "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775",
    "history": [
        {
            "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
            "timeStamp": 1629180264,
            "value": {
                "assetType": "otransaction",
                "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775",
```

```
                "token_id": "digiCurr101",
                "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df",
                "to_account_id": "",
                "transaction_type": "BULKTRANSFER",
                "amount": 20,
                "timestamp": "2021-08-17T06:04:24.000Z",
                "number_of_sub_transactions": 2,
                "holding_id": ""
            }
        }
    ],
    "sub_transactions": [
        {
            "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c4ca4238a0b923820dcc509a6f75849b",
            "history": [
                {
                    "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
                    "timeStamp": 1629180264,
                    "value": {
                        "assetType": "otransaction",
                        "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c4ca4238a0b923820dcc509a6f75849b",
                        "token_id": "digiCurr101",
                        "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df",
                        "to_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
                        "transaction_type": "TRANSFER",
                        "amount": 10,
                        "timestamp": "2021-08-17T06:04:24.000Z",
                        "number_of_sub_transactions": 0,
                        "holding_id": ""
                    }
                }
            ]
        },
        {
            "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c81e728d9d4c2f636f067f89cc14862c",
            "history": [
                {
                    "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
                    "timeStamp": 1629180264,
```

```
                    "value": {
                        "assetType": "otransaction",
                        "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775~c81e728d9d4c2f636f067f89cc14862c",
                        "token_id": "digiCurr101",
                        "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
                        "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471a
aa1210c706e",
                        "transaction_type": "TRANSFER",
                        "amount": 10,
                        "timestamp": "2021-08-17T06:04:24.000Z",
                        "number_of_sub_transactions": 0,
                        "holding_id": ""
                    }
                }
            ]
        }
    ]
}
```

**deleteHistoricalTransactions**

This method deletes older transactions from the state database.

```
@Validator(yup.date())
    public async deleteHistoricalTransactions(time_to_expiration:
Date) {
        await
this.Ctx.Auth.checkAuthorization('TRANSACTION.deleteTransactions',
'TOKEN');
        return await
this.Ctx.Transaction.deleteTransactions(time_to_expiration);
    }
```

Parameters:

- `time_to_expiration Date` – A time stamp that indicates when to delete transactions. Transaction assets that are older than the specified time will be deleted.

Return Value Example:

```
"payload": {
    "msg": "Successfuly deleted transaction older than date: Thu Aug
19 2021 11:19:36 GMT+0000 (Coordinated Universal Time).",
    "transactions": [

"otransaction~ec3366dd48b4ce2838f820f2f138648e6e55a07226713e59b411ff31b
0d21058"
```

```
    ]
}
```

### Methods for Token Behavior Management - Mintable Behavior

**`issueTokens`**

This method mints tokens, which are then owned by the caller of the method. The caller must have an account and the minter role. The number of tokens that can be minted is limited by the `max_mint_quantity` property of `mintable` behavior in the specification file. If the `max_mint_quantity` property is not specified, an unlimited number of tokens can be minted. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account with the minter role.

```
@Validator(yup.string(), yup.number().positive())
public async issueTokens(token_id: string, quantity: number) {
    const token_asset = await this.getTokenObject(token_id);
    return await this.Ctx.Token.mint(quantity, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to mint.

Returns:

- On success, a message with account details.

Return Value Example:

```
{
    "msg": "Successfully minted 1000 tokens to Account Id: \
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: user1)  ",
}
```

**`getTotalMintedTokens`**

This method returns the total number of minted tokens for a specified token. This method can be called only by a `Token Admin` or any `Org Admin` of the chaincode.

```
@Validator(yup.string())
 public async getTotalMintedTokens(token_id: string) {
     const token_asset = await this.getTokenObject(token_id);
     await this.Ctx.Auth.checkAuthorization('TOKEN.getTotalMintedTokens',
'TOKEN');
     const totalMintedTokens = await
this.Ctx.Token.getTotalMintedTokens(token_asset);
     return {
         msg: `Total minted token for Token Id: ${token_id} is $
{totalMintedTokens} tokens.`,
```

```
            quantity: totalMintedTokens
        };
    }
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON string indicating the total number of tokens.

Return Value Example:

```
{"msg":"Total minted token for Token Id: digiCurr101 is 100
tokens.","quantity":100}
```

**`getNetTokens`**

This method returns the total net number of tokens available in the system for a specified token. The net token total is the amount of tokens remaining after tokens are burned. In equation form: net tokens = total minted tokens - total burned tokens. If no tokens are burned, then the number of net tokens is equal to the total minted tokens. This method can be called only by a `Token Admin` or any `Org Admin` of the chaincode.

```
@Validator(yup.string())
public async getNetTokens(token_id: string) {
    const token_asset = await this.getTokenObject(token_id);
    await this.Ctx.Auth.checkAuthorization('TOKEN.getNetTokens',
'TOKEN');
    const netTokens = await this.Ctx.Token.getNetTokens(token_asset);
    return {
        msg: `Net supply of token for Token Id: ${token_id} is $
{netTokens} tokens.`,
        quantity: netTokens
    };
}
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON string indicating the net number of tokens.

Return Value Example:

```
{"msg":"Net supply of token for Token Id: digiCurr101 is 0
tokens.","quantity":0}
```

### Methods for Token Behavior Management - Transferable Behavior

**transferTokens**

This method transfers tokens from the caller to a specified account. The caller of the method must have an account. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.number().positive())
public async transferTokens(token_id: string, to_org_id: string, to_user_id:
string, quantity: number) {
   const token_asset = await this.getTokenObject(token_id);
   const to_account_id = await this.Ctx.Account.generateAccountId(token_id,
to_org_id, to_user_id);
   return await this.Ctx.Token.transfer(to_account_id, quantity,
token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `to_org_id: string` – The membership service provider (MSP) ID of the receiver (payee) in the current organization.
- `to_user_id: string` – The user name or email ID of the receiver.
- `quantity: number` – The number of tokens to transfer.

Returns:

- On success, a message with details for both payer and payee accounts.

Return Value Example:

```
{
    "msg": "Successfully transferred 400 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: user1) to account id:
oaccount~digicur~682bb71de419602af74e3f226345ef308445ca51010737900c112435f676
152df (Org-Id: Org1MSP, User-Id: user2) ",
}
```

**bulkTransferTokens**

This method is used to perform bulk transfer of tokens from the caller account to the accounts that are specified in the `flow` object. The quantities must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.The caller of this method must have an account already created. This method can be called only by the `AccountOwner` of the account.

```
@Validator(yup.string(), yup.array().of(yup.object()))
public async bulkTransferTokens(token_id: string, flow: object[]) {
     const token_asset = await this.getTokenObject(token_id);
     return await this.Ctx.Token.bulkTransfer(flow, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `flow : object[]` – An array of JSON objects that specify receivers and quantities.

```
[{
    "to_org_id": "Org1MSP",
    "to_user_id": "user1",
    "quantity": 10
}, {
    "to_org_id": "Org1MSP",
    "to_user_id": "user2",
    "quantity": 10
}]
```

  - `to_orgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
  - `userId: string` – The user name or email ID of the receiver.
  - `quantity: number` – The number of tokens to transfer.

Returns:

- A message indicating success.

Return Value Example:

```
{
    "msg": "Successfully transferred 20 tokens from Account
Id
'oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df' (Org-Id: Org1MSP, User-Id: admin).",
    "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
    "sub_transactions": [
        {
            "to_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
            "amount": 10
        },
        {
            "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471a
aa1210c706e",
            "amount": 10
        }
    ]
}
```

**Methods for Token Behavior Management - Holdable Behavior**

**holdTokens**

This method creates a hold on behalf of the owner of the tokens with the `to_account_id` account. A notary account is specified, which is responsible to either complete or release the hold. When the hold is created, the specified token balance from the payer is put on hold. A held balance cannot be transferred until the hold is either completed or released. The caller of this method must have an account already created. This method can be called only by the `AccountOwner` of the account.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string(), yup.string(), yup.number().positive(), yup.date())
public async holdTokens(
    token_id: string,
    operation_id: string,
    to_org_id: string,
    to_user_id: string,
    notary_org_id: string,
    notary_user_id: string,
    quantity: number,
    time_to_expiration: Date
) {
    const token_asset = await this.getTokenObject(token_id);
    const to_account_id = await this.Ctx.Account.generateAccountId(token_id,
to_org_id, to_user_id);
    const notary_account_id = await
this.Ctx.Account.generateAccountId(token_id, notary_org_id, notary_user_id);
    return await this.Ctx.Token.hold(operation_id, to_account_id,
notary_account_id, quantity, time_to_expiration, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `to_org_id: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `to_user_id: string` – The user name or email ID of the receiver.
- `notary_org_id: string` – The membership service provider (MSP) ID of the notary in the current organization.
- `notary_user_id: string` – The user name or email ID of the notary.
- `quantity: number` – The number of tokens to put on hold.
- `time_to_expiration` – The time when the hold expires. Specify `0` for a permanent hold. Otherwise use the RFC-3339 format. For example, `2021-06-02T12:46:06Z`.

Returns:

- On success, a message with the caller's account and hold details.

Return Value Example:

```
{
  "msg":"AccountId
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (Org-Id: Org1MSP , User-Id: admin) is   successfully
holding 10 tokens"
}
```

**executeHoldTokens**

This method completes a hold on a token. A quantity of tokens previously held by a token owner is transferred to a receiver. If the `quantity` value is less than the actual hold value, then the remaining amount is available again to the original owner of the tokens. This method can be called only by the `AccountOwner` ID with the `notary` role for the specified operation ID. The hold can only be completed by the notary.

```
@Validator(yup.string(), yup.string(), yup.number().positive())
public async executeHoldTokens(token_id: string, operation_id: string,
quantity: number) {
    const token_asset = await this.getTokenObject(token_id);
    return await this.Ctx.Token.executeHold(operation_id, quantity,
token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `quantity: number` – The number of on-hold tokens to transfer.

Returns:

- On success, a message with the caller's account ID and the quantity of the transaction.

Return Value Example:

```
{
 "msg":"Account Id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (Org-Id: Org1MSP, User-Id: admin) is successfully executed
'10' tokens from Operation Id 'opr_121'."
}
```

**releaseHoldTokens**

This method releases a hold on tokens. The transfer is not completed and all held tokens are available again to the original owner. This method can be called by the

`AccountOwner` ID with the `notary` role within the specified time limit or by the payer, payee, or notary after the specified time limit.

```
@Validator(yup.string(), yup.string())
public async releaseHoldTokens(token_id: string, operation_id: string) {
    const token_asset = await this.getTokenObject(token_id);
    return await this.Ctx.Token.releaseHold(operation_id, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a message indicating that the hold was released.

Return Value Example:

```
{
 "msg":"Successfully released '10' tokens from Operation Id 'opr_121' to
Account Id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: user1)."
}
```

**`getOnHoldIds`**

This method returns a list of all of the holding IDs for a specified account. This method can be called by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getOnHoldIds(token_id: string, org_id: string, user_id:
string) {
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.getOnHoldIds", "TOKEN",
{ account_id });
    return await this.Ctx.Account.getOnHoldIds(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON list of holding IDs.

Return Value Example:

```
{"msg":"Holding Ids are:
ohold~digicur~digiCurr101~opr_121","holding_ids":
["ohold~digicur~digiCurr101~opr_121"]}
```

**getOnHoldDetailsWithOperationId**
This method returns the on-hold transaction details for a specified operation ID and token. This method can be invoked by anyone.

```
@Validator(yup.string(), yup.string())
public async getOnHoldDetailsWithOperationId(token_id: string,
operation_id: string) {
    return await
this.Ctx.Hold.getOnHoldDetailsWithOperationId(token_id, operation_id);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a JSON hold object that includes the following properties:
- `holding_id` – The holding ID of the transaction.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `from_account_id` – The account ID of the current owner of the on-hold tokens.
- `to_account_id` – The account ID of the receiver.
- `notary_account_id` – The account ID of the notary.
- `token_id: string` – The ID of the saved token.
- `quantity` – The amount of tokens that are on hold for the holding ID.
- `time_to_expiration` – The duration until the hold expires.

Return Value Example:

```
{
    "assetType": "ohold",
    "holding_id": "ohold~digicur~digiCurr101~opr_121",
    "operation_id": "opr_121",
    "token_name": "digicur",
    "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
```

```
    "to_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
    "notary_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
c706e",
    "token_id": "digiCurr101",
    "quantity": 10,
    "time_to_expiration": "2022-08-01T18:30:00.000Z"
}
```

**`getOnHoldBalanceWithOperationId`**
This method returns the on-hold balance for a specified operation ID and token. This method can be invoked by anyone.

```
@Validator(yup.string(), yup.string())
public async getOnHoldBalanceWithOperationId(token_id: string, operation_id:
string) {
    return await this.Ctx.Hold.getOnHoldBalanceWithOperationId(token_id,
operation_id);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a JSON string indicating the holding balance.

Return Value Example:

```
{
    "msg": "Current Holding Balance of Operation 'opr_121' for token
'digiCurr101' is: 10",
    "holding_balance": 10
}
```

### Methods for Token Behavior Management - Burnable Behavior

**`burnTokens`**
This method deactivates, or burns, tokens from the transaction caller's account. The caller of this method must have an account and the burner role. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called by the `AccountOwner` of the account with the burner role.

```
@Validator(yup.string(), yup.number().positive())
public async burnTokens(token_id: string, quantity: number) {
```

```
    const token_asset = await this.getTokenObject(token_id);
    return await this.Ctx.Token.burn(quantity, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to burn.

Returns:

- On success, a success message with the quantity of tokens burned and the account ID.

Return Value Example:

```
{
    "msg": "Successfully burned 10 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (Org-Id: Org1MSP, User-Id: admin)"
}
```

### Custom Methods

You can use the token SDK methods to write custom methods for your business application.

To avoid double-spending, do not combine multiple async functions that operate on the same key-value pairs in the state database. Instead, use the `bulkTransferTokens` method to make multiple transfers in one method.

The following example shows how to use token SDK methods in custom methods. When the `buyTicket` method is called, it transfers 20 tokens from the caller's account to the seller's account, and returns the transaction message of the transfer.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string())
public async buyTicket(token_id: string, seller_org_id: string,
seller_user_id: string) {
    const token = await this.getTokenObject(token_id);

    /**
    * The following method
this.Ctx.Account.generateAccountId(token_id, seller_org_id,
seller_user_id) generates account id of the seller.
    */
    const seller_account_id = await
this.Ctx.Account.generateAccountId(token_id, seller_org_id,
seller_user_id);

    /**
    * The following method this.Ctx.Token.transfer(seller_account_id,
20, token) transfers the quantity 20 from caller's
    * account & to seller's account.
    */
```

```
    const transaction = await this.Ctx.Token.transfer(seller_account_id, 20,
token);

    return transaction;
}
```

If you use more than one token SDK method in a custom method, do not use methods that will affect the same key-value pairs in the state database. The following example shows the incorrect way to make multiple transfers:

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string())
public async sendTokens(token_id: string, user1_org_id: string,
user1_user_id: string, user2_org_id: string, user2_user_id: string) {
    const token = await this.getTokenObject(token_id);
    const user1_account_id = await Account.generateAccountId(token_id,
user1_org_id, user1_user_id);
    const user2_account_id = await Account.generateAccountId(token_id,
user2_org_id, user2_user_id);
    await token.transfer(user1_account_id, 20);
    await token.transfer(user2_account_id, 30);
}
```

Instead, use the `bulkTransferTokens` method to transfer to multiple accounts from the caller's account, as shown in the following code snippet.

```
bulkTransferTokens(token_id: string, flow: object[])
```

> **Note:**
>
> If you use more than one token SDK method in a custom method that might affect the same key-value pairs in the state database, enable the MVCC optimization for token chaincodes. For more information, see MVCC Optimization.

**Token SDK Methods**

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Holdable Behavior
  - Burnable Behavior

### Methods for Access Control Management

The token SDK provides an access control function. Some methods can be called only by a `Token Admin`, `Org Admin`, or `AccountOwner` of the token. You can use this feature to ensure that operations are carried out only by the intended users. Any unauthorized access results in an error. To use the access control function, import the `Authorization` class from the `../lib/auth` module.

```
import { Authorization } from '../lib/auth';
```

**addAdmin**
This method adds a user as a `Token Admin` of the token chaincode.

```
Ctx.Admin.addAdmin(org_id: string, user_id: string)
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- On success, a promise message with a JSON object that lists details for the user added as a `Token Admin` of the token chaincode. On error, a rejection with an error message.

Return Value Example:

```
{
    "msg": "Successfully added Admin (Org_Id: Org1MSP, User_Id: user1)"
}
```

**removeAdmin**
This method removes a user as a `Token Admin` of the token chaincode.

```
Ctx.Admin.removeAdmin(org_id: string, user_id: string)
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- On success, a promise message with a JSON object that lists details for the user who is no longer a `Token Admin` of the token chaincode. On error, a rejection with an error message.

Return Value Example:

```
{
    "msg": "Successfully removed Admin (Org_Id: Org1MSP, User_Id: user1)"
}
```

**getAllAdmins**

This method returns a list of all users who are a `Token Admin` of the token chaincode.

```
Ctx.Admin.getAllAdmins()
```

Parameters:

- none

Returns:

- On success, a promise with a JSON object that lists details for all users who are a `Token Admin` of the token chaincode. On error, a rejection with an error message.

Return Value Example:

```
{
    "admins": [
        {
            "orgId": "Org1MSP",
            "userId": "admin"
        }
    ]
}
```

**checkAuthorization**

Use this method to add an access control check to an operation. Certain token methods can be run only by a `Token Admin` or `AccountOwner` of the token or by the `MultipleAccountOwner` for users with multiple accounts. The access control mapping is described in the `../lib/constant.ts` file. You can modify access control by editing the `../lib/constant.ts` file. To use your own access control or to disable access control, remove the access control code from the automatically generated controller methods and custom methods.

```
export const TOKENACCESS = {
  ADMIN: {
    isUserTokenAdmin: ["Admin", "OrgAdmin"],
    addTokenAdmin: ["Admin"],
    removeTokenAdmin: ["Admin"],
    getAllAdmins: ["Admin", "OrgAdmin"],
    addOrgAdmin: ["Admin", "OrgAdminForOrgId"],
    removeOrgAdmin: ["Admin", "OrgAdminForOrgId"],
    getOrgAdmins: ["Admin", "OrgAdmin"],
  },
  TOKEN: {
    save: ["Admin"],
    getAllTokens: ["Admin", "OrgAdmin"],
```

```
    get: ["Admin", "OrgAdmin"],
    update: ["Admin"],
    getDecimals: ["Admin", "OrgAdmin"],
    getTokensByName: ["Admin", "OrgAdmin"],
    addRoleMember: ["Admin", "OrgAdminRoleCheck"],
    removeRoleMember: ["Admin", "OrgAdminRoleCheck"],
    isInRole: ["Admin", "OrgAdminForAccountId", "AccountOwner"],
    getTotalMintedTokens: ["Admin", "OrgAdmin"],
    getNetTokens: ["Admin", "OrgAdmin"],
    getTokenHistory: ["Admin", "OrgAdmin"],
  },
  ROLE: {
    getAccountsByRole: ["Admin"],
    getOrgAccountsByRole: ["Admin", "OrgAdminForOrgId"],
    getUsersByRole: ["Admin"],
    getOrgUsersByRole: ["Admin", "OrgAdminForOrgId"],
  },
  TRANSACTION: {
    deleteTransactions: ["Admin"],
  },ACCOUNT: {
    createAccount: ["Admin", "OrgAdminForOrgId"],
    associateToken: ["Admin", "OrgAdminForAccountId"],
    getAllAccounts: ["Admin"],
    getAllOrgAccounts: ["Admin", "OrgAdminForOrgId"],
    getAccountsByUser: ["Admin", "OrgAdminForOrgId",
"MultipleAccountOwner"],
    getAccount: ["Admin", "OrgAdminForAccountId", "AccountOwner"],
    history: ["Admin", "AccountOwner"],
    getAccountTransactionHistory: ["Admin", "OrgAdminForAccountId",
"AccountOwner"],
    getAccountTransactionHistoryWithFilters: ["Admin",
"OrgAdminForAccountId", "AccountOwner"],
    getSubTransactionsById: ["Admin", "TransactionInvoker"],
    getSubTransactionsByIdWithFilters: ["Admin", "TransactionInvoker"],
    getAccountBalance: ["Admin", "OrgAdminForAccountId",
"AccountOwner"],
    getAccountOnHoldBalance: ["Admin", "OrgAdminForAccountId",
"AccountOwner"],
    getOnHoldIds: ["Admin", "OrgAdminForAccountId", "AccountOwner"],
    getConversionHistory: ["Admin", "OrgAdminForAccountId",
"AccountOwner"],
  },
  ACCOUNT_STATUS: {
    get: ["Admin", "OrgAdminForAccountId", "AccountOwner"],
    history: ["Admin", "OrgAdminForAccountId", "AccountOwner"],
    activateAccount: ["Admin", "OrgAdminForOrgId"],
    suspendAccount: ["Admin", "OrgAdminForOrgId"],
    deleteAccount: ["Admin", "OrgAdminForOrgId"],
  },
  TOKEN_CONVERSION: {
    initializeExchangePoolUser: ["Admin"],
    addConversionRate: ["Admin"],
    updateConversionRate: ["Admin"],
```

```
    getConversionRate: ["Admin", "OrgAdmin", "AnyAccountOwner"],
    getConversionRateHistory: ["Admin", "OrgAdmin", "AnyAccountOwner"],
    tokenConversion: ["Admin", "AnyAccountOwner"],
    getExchangePoolUser: ["Admin"],
  },
}
```

```
await this.Ctx.Auth.checkAuthorization(<parameters>);
```

Parameters:

- `classFuncName: string` – The map value between the class and methods as described in the `../lib/constant.ts` file.
- `...args` – A variable argument where `args[0]` takes the constant `'TOKEN'` and `args[1]` takes the `account_id` to add an access control check for an `AccountOwner`. To add an access control check for a `MultipleAccountOwner`, `args[1]` takes the `org_id` and `args[2]` takes the `user_id`.

Returns:

- On success, a promise. On error, a rejection with an error message.

**`isUserTokenAdmin`**
This method returns the Boolean value `true` if the caller of the function is a `Token Admin`. Otherwise the method returns `false`.

```
Ctx.Auth.isUserTokenAdmin()
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- A Boolean response and an error message if an error is encountered.

**`addOrgAdmin`**
This method adds a user as an `Org Admin` of the organization.

```
Ctx.Admin.addOrgAdmin(org_id, user_id)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully added Org Admin (Org_Id: Org1MSP, User_Id:
orgAdmin)"
}
```

**removeOrgAdmin**
This method removes a user as an `Org Admin` of the organization.

```
Ctx.Admin.removeOrgAdmin(org_id, user_id)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as an `Org Admin` of the organization.

Return Value Example:

```
{
  "msg": "Successfully removed Org Admin (Org_Id Org1MSP User_Id
orgAdmin)"
}
```

**getOrgAdmins**
This method returns a list of all users who are an `Org Admin` of an organization.

```
Ctx.Admin.getAllOrgAdmins()
```

Parameters:

- none

Returns:

- On success, an array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{
    "admins": [
        {
```

```
            "org_id": "Org1MSP",
            "user_id": "orgadmin"
        },
        {
            "org_id": "Org1MSP",
            "user_id": "orgadmin1"
        },
        {
            "org_id": "Org1MSP",
            "user_id": "orgadmin2"
        }
    ]
}
```

### Methods for Token Configuration Management

**`getTokenDecimals`**
This method returns the number of decimal places available for a fractional token. If the `divisible` behavior is not specified, then the default value is 0.

```
Ctx.Token.getTokenDecimals(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, the decimal places of the token, in the number data type. On error, it returns with an error message.

Return Value Example:

```
1
```

**`getAllTokens`**
This method returns all the token assets saved in the state database. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.Token.getAllTokens()
```

Parameters:

- none

Returns:

- On success, it returns a promise with all the token assets. On error, it returns an error message.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"98e0a0a115803d25b843d630e6b23c435a192a03eb0a301fc9375f05da49a8b2",
        "payload": [
            {
                "key": "token1",
                "valueJson": {
                    "assetType": "otoken",
                    "token_id": "token1",
                    "token_name": "vtok",
                    "token_type": "fungible",
                    "behaviours": [
                        "divisible",
                        "mintable",
                        "transferable",
                        "burnable",
                        "holdable",
                        "roles"
                    ],
                    "roles": {
                        "burner_role_name": "burner",
                        "notary_role_name": "notary"
                    },
                    "mintable": {
                        "max_mint_quantity": 0
                    },
                    "divisible": {
                        "decimal": 1
                    }
                }
            }
        ],
        "encode": "JSON"
    }
}
```

**`getTokensByName`**

This method returns all the token assets with the specified name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.Token.getTokensByName(token_name: string)
```

Parameters:

- `token_name: string` – The name of the token, which corresponds to the `Token_name` property of the model. The value is the class name of the token.

Returns:

- It returns an array of all of the token assets of the specified name, in JSON format.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"98e0a0a115803d25b843d630e6b23c435a192a03eb0a301fc9375f05da49a8b2",
        "payload": [
            {
                "key": "token1",
                "valueJson": {
                    "assetType": "otoken",
                    "token_id": "token1",
                    "token_name": "vtok",
                    "token_type": "fungible",
                    "behaviours": [
                        "divisible",
                        "mintable",
                        "transferable",
                        "burnable",
                        "holdable",
                        "roles"
                    ],
                    "roles": {
                        "burner_role_name": "burner",
                        "notary_role_name": "notary"
                    },
                    "mintable": {
                        "max_mint_quantity": 0
                    },
                    "divisible": {
                        "decimal": 1
                    }
                }
            }
        ],
        "encode": "JSON"
    }
}
```

**`get`**
This method returns a token object if it is present in the state database.

```
Ctx.Token.get(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token to return.

Returns:

- On success, a promise with the JSON representation of the token. On error, a rejection with an error message.

Return Value Example:

```
{
    "assetType": "otoken",
    "token_id": "token1",
    "token_name": "account",
    "token_desc": "Token 1",
    "token_type": "fungible",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "holdable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner",
        "notary_role_name": "notary"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "divisible": {
        "decimal": 1
    },
    "token_to_currency_ratio": 2,
    "currency_representation": "EURO"
}
```

**`isTokenType`**
This method indicates whether a token asset exists with the specified ID.

```
Ctx.Token.isTokenType(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token to check.

Returns:

- On success, a promise with `true` if a token asset exists with the specified ID. On error, a rejection with an error message.

Return Value Example:

```
true
```

**save**

This method creates a token and saves its properties in the state database.

```
Ctx.Token.save(token: <Instance of Token Class>,extraMetadata?:any)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Return Value Example:

```
{
   "assetType":"otoken",
   "token_id":"digiCurr101",
   "token_name":"digicur",
   "token_type":"fungible",
   "behaviors":[
      "divisible",
      "mintable",
      "transferable",
      "burnable",
      "roles"
   ],
   "roles":{
      "minter_role_name":"minter"
   },
   "mintable":{
      "max_mint_quantity":1000
   },
   "divisible":{
      "decimal":2
   },
   "currency_name":"DOLLAR",
   "token_to_currency_ratio":1
}
```

**update**

This method updates token properties. After a token asset is created, you update only the `token_desc` value and its custom properties.

```
Ctx.Token.update(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Return Value Example:

```
{
   "assetType":"otoken",
   "token_id":"digiCurr101",
   "token_name":"digicur",
   "token_desc":"Digital Currency equiv of dollar",
   "token_type":"fungible",
   "behaviors":[
      "divisible",
      "mintable",
      "transferable",
      "burnable",
      "roles"
   ],
   "roles":{
      "minter_role_name":"minter"
   },
   "mintable":{
      "max_mint_quantity":1000
   },
   "divisible":{
      "decimal":2
   },
   "currency_name":"DOLLAR",
   "token_to_currency_ratio":1
}
```

**`getByRange`**

This method calls the fabric `getStateByRange` method internally. Even though any asset with the given ID is returned from the ledger, this method casts the asset into the caller Asset type.

```
<Token ClassCtx.Token.getByRange(start_token_id: string, end_token_id:
string, token_class_reference?: <Instance of Token Class> )
```

Parameters:

- `startId: string` – The starting key of the range. This key is included in the range.
- `endId: string` – The end key of the range. This key is excluded from the range.
- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with an array of `<Token Class>`. On error, a rejection with an error message.

Example:

```
@validator(yup.string(), yup.string())
public async getDigiCurrGetByRange(start_token_id: string, end_token_id:
string) {
   return await this.Ctx.Token.getByRange(start_token_id, end_token_id,
DigiCurr);
}
```

Return Value Example:

```
[
    {
        "assetType": "otoken",
        "token_id": "token1",
        "token_name": "digicur",
        "token_desc": "Token 1",
        "token_type": "fungible",
        "behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "holdable",
            "roles"
        ],
        "roles": {
            "minter_role_name": "minter",
            "burner_role_name": "burner",
            "notary_role_name": "notary"
        },
        "mintable": {
            "max_mint_quantity": 20000
        },
        "divisible": {
            "decimal": 0
        },
        "token_to_currency_ratio": 1.5,
        "currency_representation": "USD"
    },
    {
        "assetType": "otoken",
        "token_id": "token2",
        "token_name": "digicur",
        "token_desc": "Token2",
        "token_type": "fungible",
        "behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "holdable",
```

```
            "roles"
        ],
        "roles": {
            "minter_role_name": "minter",
            "burner_role_name": "burner",
            "notary_role_name": "notary"
        },
        "mintable": {
            "max_mint_quantity": 20000
        },
        "divisible": {
            "decimal": 0
        },
        "token_to_currency_ratio": 1,
        "currency_representation": "EURO"
    }
]
```

**history**

This method returns history for the specified token.

```
Ctx.Token.history(tokenId)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a promise with an array of the account history details for the specified token. On error, a rejection with an error message.

Return Value Example:

```
[
    {
        "trxId":
"0d75f09446a60088afb948c6aca046e261fddcd43df416076201cdc5565f1a35",
        "timeStamp": "2023-09-01T16:48:41.000Z",
        "value": {
            "assetType": "otoken",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "token_desc": "updatedDesc",
            "token_standard": "ttf+",
            "token_type": "fungible",
            "token_unit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
```

```
            ],
            "roles": {
                "minter_role_name": "minter"
            },
            "mintable": {
                "max_mint_quantity": 1000
            },
            "divisible": {
                "decimal": 2
            }
        }
    },
    {
        "trxId":
"3666344878b043b65d5b821cc79c042ba52aec467618800df5cf14eac69f72fa",
        "timeStamp": "2023-08-31T20:24:55.000Z",
        "value": {
            "assetType": "otoken",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "token_standard": "ttf+",
            "token_type": "fungible",
            "token_unit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter"
            },
            "mintable": {
                "max_mint_quantity": 1000
            },
            "divisible": {
                "decimal": 2
            }
        }
    }
]
```


### Methods for Account Management

**`getCallerAccountId`**
This method returns the account ID of the caller.

```
Ctx.Account.getCallerAccountId(token_id: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a promise with the caller account ID. On error, a rejection with an error message.

Return Value Example:

```
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f
```

**`generateAccountId`**

This method returns an account ID, which is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`org_id`) of the user in the current network organization and the unique token ID (`token_id`).

```
Ctx.Account.generateAccountId(token_id: string, org_id: string,
user_id: string)
```

Parameters:

- `tokenId: string` – The ID of the token.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a promise with the generated account ID. On error, a rejection with an error message.

Return Value Example:

```
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f
```

**`createAccount`**

This method creates an account for a specified user and token. Every user who has tokens at any point must have an account. Accounts track a user's balance, on-hold balance, and transaction history. An account ID is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`org_id`) of the user in the current

network organization. This method can be called only by the `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
this.Ctx.Account.createAccount(org_id: string, user_id: string, token_type:
string)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.
- `token_type: string` – The type of the token, which must be `fungible`.

Returns:

- On success, the new account object in JSON format.

Return Value Example:

```
{
  "assetType": "oaccount",
  "bapAccountVersion": 0,
  "account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",
  "user_id": "admin",
  "org_id": "Org1MSP",
  "token_type": "fungible",
  "token_id": "",
  "token_name": "",
  "balance": 0,
  "onhold_balance": 0
}
```

**`associateTokenToAccount`**

This method associates a fungible token with an account. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the relevant organization.

```
async associateTokenToAccount(account_id: string, token_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.
- `token_id: string` – The ID of the token.

Returns:

- On success, a JSON object of the updated account.

Return Value Example:

```
{
    "assetType": "oaccount",
    "bapAccountVersion": 0,
    "account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",
    "user_id": "admin",
    "org_id": "Org1MSP",
    "token_type": "fungible",
    "token_id": "fungible",
    "token_name": "fiatmoneytok",
    "balance": 0,
    "onhold_balance": 0
}
```

**`getAccountWithStatus`**
This method returns account details for a specified account, including account status.

```
Ctx.Account.getAccountWithStatus(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with the account details. On error, a rejection with an error message.

Return Value Example:

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "account_id":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
  "user_id": "idcqa",
  "org_id": "appdev",
  "token_type": "fungible",
  "token_id": "t1",
  "token_name": "obptok",
  "balance": 0,
  "onhold_balance": 0
}
```

**getAccount**
This method returns account details for a specified account.

```
Ctx.Account.getAccount(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with the account details. On error, a rejection with an error message.

Return Value Example:

```
{
   "assetType":"oaccount",
   "bapAccountVersion": 0,

"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d
546d64f368642f622f",
   "user_id":"user1",
   "org_id":"Org1MSP",
   "token_id":"digiCurr101",
   "token_name":"digicur",
   "balance":0,
   "onhold_balance":0
}
```

**history**
This method returns an array of the account history details for a specified account.

```
Ctx.Account.history(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with the array of account history details. On error, a rejection with an error message. The return value is the same as the getAccountHistory method.

Return Value Example:

```
[
    {

"trxId":"2gsdh17fff222467e5667be042e33ce18e804b3e065cca15de306f837e416d7c3e",
      "timeStamp":1629718288,
      "value":{
         "assetType":"oaccount",
```

```
"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62c
ba5c3d546d64f368642f622f",
         "user_id":"user1",
         "org_id":"Org1MSP",
         "token_id":"digiCurr101",
         "token_name":"digicur",
         "balance":100,
         "onhold_balance":0,
         "bapAccountVersion": 1
   },
   {

"trxId":"9fd07fff222467e5667be042e33ce18e804b3e065cca15de306f837e416d7c
3e",
      "timeStamp":1629718288,
      "value":{
         "assetType":"oaccount",

"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62c
ba5c3d546d64f368642f622f",
         "user_id":"user1",
         "org_id":"Org1MSP",
         "token_id":"digiCurr101",
         "token_name":"digicur",
         "balance":0,
         "onhold_balance":0,
         "bapAccountVersion": 0
      }
   }
]
```


**getAccountOnHoldBalance**

This method returns the on-hold balance for a specified account.

```
Ctx.Account.getAccountOnHoldBalance(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with a JSON object that shows the on-hold balance for the specified account. On error, a rejection with an error message.

Return Value Example:


```
{
   "holding_balance":0,
   "msg":"Total Holding Balance of Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
```

```
68642f622f (org_id: Org1MSP, user_id: user1) is 0"
}
```

**getAllAccounts**

This method returns a list of all accounts. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.Account.getAllAccounts()
```

Parameters:

- none

Returns:

- On success, a promise with a JSON object that lists all accounts. On error, a rejection with an error message.

Return Value Example:

```
[
            {
                "key":
"oaccount~digicur~2e2ef3375ae347cbd7b4d3d7be5cece803f9c36a184aaf2b8d332c5d2dc
ead52",
                "valueJson": {
                    "assetType": "oaccount",
                    "account_id":
"oaccount~digicur~2e2ef3375ae347cbd7b4d3d7be5cece803f9c36a184aaf2b8d332c5d2dc
ead52",
                    "user_id": "admin",
                    "org_id": "Org1MSP",
                    "token_id": "digiCurr101",
                    "token_name": "digicur",
                    "bapAccountVersion": 0,
                    "balance": 0,
                    "onhold_balance": 0
                }
            },
            {
                "key":
"oaccount~digicur~30080c7e5ba94035af57fbbccbbb495e92515e4b2b3dbcd476eb1c0343e
4da65",
                "valueJson": {
                    "assetType": "oaccount",
                    "account_id":
"oaccount~digicur~30080c7e5ba94035af57fbbccbbb495e92515e4b2b3dbcd476eb1c0343e
4da65",
                    "bapAccountVersion": 0,
                    "user_id": "user1",
                    "org_id": "Org1MSP",
                    "token_id": "digiCurr101",
                    "token_name": "digicur",
```

```
                    "balance": 0,
                    "onhold_balance": 0
                }
            },
            {
                "key":
"oaccount~digicur~cbde438258cb01a82f71a9a9f8029243c40c6d836a50543212052
9c2b3c2ff0c",
                "valueJson": {
                    "assetType": "oaccount",
                    "account_id":
"oaccount~digicur~cbde438258cb01a82f71a9a9f8029243c40c6d836a50543212052
9c2b3c2ff0c",
                    "bapAccountVersion": 0,
                    "user_id": "user2",
                    "org_id": "Org1MSP",
                    "token_id": "digiCurr101",
                    "token_name": "digicur",
                    "balance": 0,
                    "onhold_balance": 0
                }
            },
            {
                "key":
"oaccount~digicur~ecbc3aefcc562d3049c988717940195b30297e95012b7824bbd33
a57ca50a626",
                "valueJson": {
                    "assetType": "oaccount",
                    "account_id":
"oaccount~digicur~ecbc3aefcc562d3049c988717940195b30297e95012b7824bbd33
a57ca50a626",
                    "bapAccountVersion": 0,
                    "user_id": "user3",
                    "org_id": "Org1MSP",
                    "token_id": "digiCurr101",
                    "token_name": "digicur",
                    "balance": 500,
                    "onhold_balance": 0
                }
            }
        ]
```


**getUserByAccountId**

This method returns the user details for a specified account.

```
Ctx.Account.getUserByAccountId(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with a JSON object that includes three properties:
  - `user_id` – The user name or email ID of the user.
  - `org_id` – The membership service provider (MSP) ID of the user in the current network organization.
  - `token_id` – The ID of the token.
- On error, a rejection with an error message.

Return Value Example:

```
{
   "token_id": "digiCurr101",
   "user_id": "user1",
   "org_id": "Org1MSP"
}
```

**`getAccountBalance`**
This method returns the account balance for a specified account.

```
Ctx.Account.getAccountBalance(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise message with a JSON object that includes two properties:
  - `msg` – A message showing the current balance.
  - `user_balance` – The numeric value of the current balance.
- On error, a rejection with an error message.

Return Value Example:

```
{
    "msg": "Current Balance is: 200",
    "user_balance": 200
}
```

**`getAllOrgAccounts`**
This method returns a list of all token accounts that belong to a specified organization.

```
Ctx.Account.getAllOrgAccounts(org_id: string)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts for the specified organization.

Return Value Example:


```
[
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
        "valueJson": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
            "user_id": "idcqa",
            "org_id": "appdev",
            "token_type": "fungible",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "balance": 0,
            "onhold_balance": 0
        }
    },
    {
        "key":
"oaccount~620fcf5deb5fd5a65c0b5b10fda129de0f629ccd232c5891c130e24a574df
50a",
        "valueJson": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~620fcf5deb5fd5a65c0b5b10fda129de0f629ccd232c5891c130e24a574df
50a",
            "user_id": "example_minter",
            "org_id": "appdev",
            "token_type": "fungible",
            "token_id": "token",
            "token_name": "fiatmoneytok",
            "balance": 0,
            "onhold_balance": 0
        }
    }
]
```


### Methods for Role Management

**`addRoleMember`**

This method adds a role to a specified user and token.

```
Ctx.Token.addRoleMember(role: string, account_id: string, token:
<Instance of Token Class>)
```

Parameters:

- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `account_id: number` – The account ID to add the role to.
- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
    "msg":"Successfully added role minter to
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id :          Org1MSP, user_id : user1)"
}
```

**`removeRoleMember`**
This method removes a role from a specified user and token.

```
Ctx.Token.removeRoleMember(role: string, account_id: string, token:
<Instance of Token Class>)
```

Parameters:

- `role: string` – The name of the role to remove from to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `account_id: number` – The account ID to remove the role from.
- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
  "msg":"successfully removed member_id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1) from role minter"
}
```

**getAccountsByRole**
This method returns a list of all accounts for a specified role and token.

```
Ctx.Role.getAccountsByRole(token_id: string, role: string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a promise with a JSON object that lists all accounts for the specified role and token. On error, a rejection with an error message.

Return Value Example:

```
{
    "accounts": [

"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",

"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f"
    ]
}
```

**getAccountsByUser**
This method returns a list of all account IDs for a specified user.

```
async getAccountsByUser(org_id: string, user_id: string)
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{"accounts":
["oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64
f368642f622f"]}
```

**getUsersByRole**
This method returns a list of all users for a specified role and token.

```
Ctx.Role.getUsersByRole(token_id: string, role: string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a promise with a JSON object that lists all users for the specified role and token. On error, a rejection with an error message.

Return Value Example:

```
{
   "users":[
      {
         "token_id":"digiCurr101",
         "user_id":"user1",
         "org_id":"Org1MSP"
      }
   ]
}
```

**isInRole**
This method indicates whether a user and token has a specified role.

```
Ctx.Token.isInRole(role: string, account_id: string, token: <Instance of
Token Class>)
```

Parameters:

- `role: string` – The name of the role to check.
- `account_id: number` – The account ID to check.
- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with `true` if the user has the role, and `false` if the user does not have the role. On error, a rejection with an error message.

Return Value Example:

```
{"result":"true"}
```

**roleCheck**
This method checks if the provided account ID is a member of any role.

```
Ctx.Token.roleCheck(account_id: string, token: <Instance of Token
Class>)
```

Parameters:

- `account_id: string` – The account ID to check.
- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- If the account ID is part of any role, it returns `true`. Otherwise, it returns `false`.

Return Value Example:

```
{ result: true }
```

**getOrgUsersByRole**
This method returns information about all users that have a specified role in a specified organization.

```
Ctx.Role.getOrgUsersByRole(token_id: string, role: string, org_id:
string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all users with the specified role in the specified organization.

Return Value Example:

```
{
    "users": [
        {
            "token_id": "token",
            "user_id": "admin",
            "org_id": "Org1MSP"
        },
        {
            "token_id": "token",
            "user_id": "orgAdmin",
            "org_id": "Org1MSP"
        }
```

```
    ]
}
```

**getOrgAccountsByRole**
This method returns information about all accounts that have a specified role in a specified organization.

```
Ctx.Role.getOrgAccountsByRole(token_id: string, role: string, org_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts with the specified role in the specified organization.

Return Value Example:

```
{
    "accounts": [

"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",

"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a850"
    ]
}
```

### Methods for Transaction History Management

**getTransactionById**
This method returns the history of a `Transaction` asset.

```
async getTransactionById(transaction_id: string)
```

Parameters:

- `transaction_id: string` – The ID of the transaction asset.

Returns:

- On success, the transaction asset history.

Return Value Example:

```
{
    "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5",
```

```
    "history": [
        {
            "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
            "timeStamp": 1629180264,
            "value": {
                "assetType": "otransaction",
                "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775",
                "token_id": "digiCurr101",
                "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
                "to_account_id": "",
                "transaction_type": "BULKTRANSFER",
                "amount": 20,
                "timestamp": "2021-08-17T06:04:24.000Z",
                "number_of_sub_transactions": 2,
                "holding_id": ""
            }
        }
    ],
    "sub_transactions": [
        {
            "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775~c4ca4238a0b923820dcc509a6f75849b",
            "history": [
                {
                    "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
                    "timeStamp": 1629180264,
                    "value": {
                        "assetType": "otransaction",
                        "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8
e0fc775~c4ca4238a0b923820dcc509a6f75849b",
                        "token_id": "digiCurr101",
                        "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
                        "to_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
                        "transaction_type": "TRANSFER",
                        "amount": 10,
                        "timestamp": "2021-08-17T06:04:24.000Z",
                        "number_of_sub_transactions": 0,
                        "holding_id": ""
                    }
                }
            ]
```

```
        },
        {
            "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c81e728d9d4c2f636f067f89cc14862c",
            "history": [
                {
                    "trxId":
"68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc775",
                    "timeStamp": 1629180264,
                    "value": {
                        "assetType": "otransaction",
                        "transaction_id":
"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965e63a9bbf1d8e0fc77
5~c81e728d9d4c2f636f067f89cc14862c",
                        "token_id": "digiCurr101",
                        "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df",
                        "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
c706e",
                        "transaction_type": "TRANSFER",
                        "amount": 10,
                        "timestamp": "2021-08-17T06:04:24.000Z",
                        "number_of_sub_transactions": 0,
                        "holding_id": ""
                    }
                }
            ]
        }
    ]
}
```

**deleteHistoricalTransactions**

This method returns an array of the transaction history details for a specified account.

```
async deleteHistoricalTransactions(time_to_expiration: Date)
```

Parameters:

- `time_to_expiration: Date` – A time stamp that indicates when to delete transactions. Transaction assets that are older than the specified time will be deleted..

Returns:

- The return value is the same as the getAccountTransactionHistory method.
- On success, a promise with the array of account transaction objects:
  - `transaction_id` – The ID of the transaction.
  - `transacted_account` – The account with which the transaction took place.

  - `transaction_type` – The type of transaction.
  - `transacted_amount` – The amount of the transaction.
  - `timestamp` – The time of the transaction.
  - `balance` – The account balance at the time of the transaction.
  - `onhold_balance` – The on-hold balance at the time of the transaction.
  - `sub_transactions` – For bulk transfers only, a list of transactions that are part of a bulk transfer.
  - `holding_id` – A unique identifier returned by the `holdTokens` method.
  - `token_id` – The ID of the token.
- On error, a rejection with an error message.

Return Value Example:

```
"payload": {
            "msg": "Successfuly deleted transaction older than date:
Thu Aug 19 2021 11:19:36 GMT+0000 (Coordinated Universal Time).",
            "transactions": [

"otransaction~ec3366dd48b4ce2838f820f2f138648e6e55a07226713e59b411ff31b
0d21058"
            ]
        }
```

**`getAccountTransactionHistory`**
This method returns an array of the transaction history details for a specified account.

```
Ctx.Account.getAccountTransactionHistory(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- The return value is the same as the getAccountTransactionHistory method.
- On success, a promise with the array of account transaction objects:
  - `transaction_id` – The ID of the transaction.
  - `transacted_account` – The account with which the transaction took place.
  - `transaction_type` – The type of transaction.
  - `transacted_amount` – The amount of the transaction.
  - `timestamp` – The time of the transaction.
  - `balance` – The account balance at the time of the transaction.

  - `onhold_balance` – The on-hold balance at the time of the transaction.
  - `sub_transactions` – For bulk transfers only, a list of transactions that are part of a bulk transfer.
  - `holding_id` – A unique identifier returned by the `holdTokens` method.
  - `token_id` – The ID of the token.
- On error, a rejection with an error message.

Return Value Example:

```
[
   {

"transaction_id":"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965
e63a9bbf1d8e0fc775",
      "transacted_amount":20,
      "timestamp":"2021-08-17T06:04:24.000Z",
      "balance":60,
      "onhold_balance":0,
      "token_id":"digiCurr101",
      "transaction_type":"BULKTRANSFER",
      "sub_transactions":[
         {

"transacted_account":"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca
51010737900c1d85f0376152df",
            "transaction_type":"CREDIT",

"transaction_id":"otransaction~68f46c90d0d8d6b93d827e6b9e0152b4845e6e42a61965
e63a9bbf1d8e0fc775~c4ca4238a0b923820dcc509a6f75849b",
            "transacted_amount":10
         }
      ]
   },
   {

"transaction_id":"otransaction~757864d5369bd0539d044caeb3bb4898db310fd7aa740f
45a9938771903d43da",
      "transacted_amount":50,
      "timestamp":"2021-08-17T06:02:44.000Z",
      "balance":50,
      "onhold_balance":0,
      "token_id":"digiCurr101",

"transacted_account":"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca
51010737900c1d85f0376152df",
      "transaction_type":"CREDIT"
   }
]
```

**getAccountTransactionHistoryWithFilters**

This method returns an array of the transaction history details for a specified account. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
await
this.Ctx.Account.getAccountTransactionHistoryWithFilters(account_id:
string, filters?: Filters)
```

Parameters:

- `account_id: string` – The ID of the account.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke getAccountTransactionHistoryWithFilters 'token1'
'appbuilder12' 'user_minter'
'{"PageSize":10,"Bookmark":"1","StartTime":"2022-01-25T17:41:42Z","EndTim
e":"2022-01-25T17:59:10Z"}'
```

```
[
    {
        "transaction_id":
"otransaction~672897b5a4fa78b421c000e4d6d4f71f3d46529bfbb5b4be10bf5471d
c35ce89",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:46:04.000Z",
        "token_id": "token1",
        "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c27488
3b8",
        "transaction_type": "DEBIT",
        "balance": 90,
        "onhold_balance": 0
    },
    {
        "transaction_id":
"otransaction~467bb67a33aaffca4487f33dcd46c9844efdb5421a2e7b6aa2d53152e
b2c6d85",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:45:47.000Z",
        "token_id": "token1",
        "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c
248",
        "transaction_type": "DEBIT",
        "balance": 95,
```

```
        "onhold_balance": 0
    },
    {
        "transaction_id":
"otransaction~c6d56ce54a9bbe24597d1d10448e39316dc6f16328bf3c5b0c8ef10e1dfeb39
7",
        "transacted_amount": 100,
        "timestamp": "2022-04-20T15:44:26.000Z",
        "token_id": "token1",
        "transacted_account":
"oaccount~deb5fb0906c40506f6c2d00c573b774e01a53dd91499e651d92ac4778b6add6a",
        "transaction_type": "MINT",
        "balance": 100,
        "onhold_balance": 0
    }
]
```

**getSubTransactionHistory**

This method returns an array of the transaction history details for a specified transaction.

```
await this.Ctx.Account.getSubTransactionHistory(transaction_id)
```

Parameters:

- `transaction_id: string` – The ID of the bulk transfer transaction.

Example:

```
ochain invoke GetAccountSubTransactionHistory
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9b'
```

```
[
    {
        "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c274883b8",
        "transaction_type": "DEBIT",
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d7fe6cb
8~c81e728d9d4c2f636f067f89cc14862c",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:52:21.000Z",
        "token_id": "token1",
        "balance": 80,
        "onhold_balance": 0
    },
    {
        "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c248",
        "transaction_type": "DEBIT",
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d7fe6cb
8~c4ca4238a0b923820dcc509a6f75849b",
        "transacted_amount": 5,
```

```
        "timestamp": "2022-04-20T15:52:21.000Z",
        "token_id": "token1",
        "balance": 85,
        "onhold_balance": 0
    }
]
```

**getSubTransactionHistoryWithFilters**

This method returns an array of the subtransaction history details for a specified transaction.

```
await
this.Ctx.Account.getSubTransactionHistoryWithFilters(transaction_id:
string, filters?: SubTransactionFilters)
```

Parameters:

- `transaction_id: string` – The ID of the bulk transfer transaction.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetAccountSubTransactionHistoryWithFilters
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f8
64b9b' '{"PageSize":10,"Bookmark":"1"}'
```

```
[
    {
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d
7fe6cb8~c81e728d9d4c2f636f067f89cc14862c",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:52:21.000Z",
        "token_id": "token1",
        "transacted_account":
"oaccount~16c38d804413ebabf416360d374f76c973d4e71c74adfde73cc40c7c27488
3b8",
        "transaction_type": "DEBIT",
        "balance": 80,
        "onhold_balance": 0
    },
    {
        "transaction_id":
"otransaction~6e0f8fe4a6430322170b9c619b04b6c9f1c8d257923f611b866bdf69d
7fe6cb8~c4ca4238a0b923820dcc509a6f75849b",
        "transacted_amount": 5,
        "timestamp": "2022-04-20T15:52:21.000Z",
```

```
        "token_id": "token1",
        "transacted_account":
"oaccount~fbf95683b21bbc91a22205819ac1e2e9c90355d536821ed3fe22b7d23915c248",
        "transaction_type": "DEBIT",
        "balance": 85,
        "onhold_balance": 0
    }
]
```

### Token Behavior Management

The token lifecycle management methods are based on the the standards of the Token Taxonomy Framework. To use the token lifecycle methods, import the `Token` class from the `../lib/token` module.

```
import { Token } from '../lib/token';
```

### Methods for Token Behavior Management - Mintable Behavior

**`mint`**
This method mints a quantity of tokens, which are then owned by the caller of the method. The caller must have an account and the minter role. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.

```
Ctx.Token.mint(quantity: number, token: <Instance of Token Class>)
```

Parameters:

- `quantity: number` – The total number of tokens to mint.
- `token: <Instance of Token Class>` – The token asset to mint.

Returns:

- On success, a promise with a success message and `toAccount` details. On error, a rejection with an error message.

Return Value Example:

```
{
  "msg":"Successfully minted 1000 tokens to Account Id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: admin)"
}
```

**`getTotalMintedTokens`**
This method returns the total number of tokens minted.

```
Ctx.Token.getTotalMintedTokens(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, the quantity of minted tokens, in the number data type. On error, it returns with an error message.

Return Value Example:

```
4000
```

**`getNetTokens`**
This method returns the net quantity of tokens that are available in the system. The net tokens are the amount of tokens remaining after tokens are burned. In equation form: net tokens = total minted tokens - total burned tokens. If no tokens are burned, then the number of net tokens is equal to the total minted tokens.

```
Ctx.Token.getNetTokens(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, the quantity of net tokens, in the number data type. On error, it returns with an error message.

Return Value Example:

```
2000
```

**`getMaxMintQuantity`**
This method returns the maximum mintable quantity for a token. If the `max_mint_quantity` behavior is not specified, then the default value is 0, which allows any number of tokens to be minted.

```
Ctx.Token.getMaxMintQuantity(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, the maximum mintable quantity of the token, in the number data type. On error, it returns with an error message.

Return Value Example:

```
20000
```

### Methods for Token Behavior Management - Transferable Behavior

**transfer**

This method transfers tokens from the transaction caller to the `to_account_id` account. The caller of this method must have an account and the quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.

```
Ctx.Token.transfer(to_account_id: string, quantity: number, token: <Instance
of Token Class>)
```

Parameters:

- `to_account_id: string` – The account ID to receive the tokens.
- `quantity: number` – The total number of tokens to transfer.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
 "msg":"Successfully transferred 50 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: admin) to account id:
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (Org-Id: Org1MSP, User-Id: user1)"
}
```

**bulkTransfer**

This method is used to perform bulk transfer of tokens from the caller account to the accounts that are specified in the `flow` object. The caller of this method must have an account already created.

```
Ctx.Token.bulkTransfer(flow: object[], token: <Instance of Token Class>)
```

Parameters:

- `flow: object[]` – An array of JSON objects specifying the receiver details and quantity. The transfer quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. For example:

```
[{
    "to_org_id": "Org1MSP",
    "to_user_id": "user1",
    "quantity": 10
}, {
    "to_org_id": "Org1MSP",
    "to_user_id": "user2",
    "quantity": 10
}]
```

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with a success message and account information. On error, a rejection with an error message.

Return Value Example:

```
{
    "from_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "msg": "Successfully transferred 2 tokens from Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (Org-Id: Org1MSP, User-Id: user1)",
    "sub_transactions": [
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471a
aa1210c706e"
        },
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df"
        }
    ]
}
```

### Methods for Token Behavior Management - Holdable Behavior

**`hold`**

This method creates a hold on behalf of the owner of the tokens with the `to_account_id` account. A notary account is specified, which is responsible to either complete or release the hold. When the hold is created, the specified token balance from the payer is put on hold. A held balance cannot be transferred until the hold is either completed or released. The caller of this method must have an account already created.

```
Ctx.Token.hold(operation_id: string, to_account_id: string,
notary_account_id: string, quantity: number, time_to_expiration: Date,
token: <Instance of Token Class>)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `to_account_id: string` – The ID of the account to receive the tokens.
- `notary__account_id: string` – The ID of the notary account.

- `quantity: number` – The total number of tokens to put on hold.
- `time_to_expiration: Date` – The duration until the hold expires. Specify `0` for a permanent hold. Otherwise use the RFC-3339 format. For example, `2021-06-02T12`.
- `token: <Instance of Token Class>` – The token asset to hold.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
 "msg": "account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (org_id : Org1MSP, user_id : user1) is successfully holding 10 tokens",
}
```

**executeHold**

This method completes a hold on tokens, transferring the specified quantity of tokens previously on hold to the receiver. If the `quantity` value is less than the actual hold value, then the remaining amount is available again to the original owner of the tokens. This method can be called only by the `AccountOwner` ID with the `notary` role for the specified operation ID. The hold can only be completed by the notary.

```
Ctx.Token.executeHold(operation_id: string, quantity: number, token:
<Instance of Token Class>)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `quantity: number` – The total number of tokens to complete a hold on.
- `token: <Instance of Token Class>` – The token asset to complete a hold on.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
 "msg": "user with accountId:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (org_id : Org1MSP, user_id : user1) has successfully executed 5
tokens(digiCurr101) from the hold with Operation Id opr_121",
}
```

**releaseHold**
This method releases a hold on tokens. The transfer is not completed and all held tokens are available again to the original owner. This method can be called by the `AccountOwner` ID with the `notary` role within the specified time limit or by the payer, payee, or notary after the specified time limit.

```
Ctx.Token.releaseHold(operation_id: string, token: <Instance of Token
Class>)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `token: <Instance of Token Class>` – The token asset to release a hold on.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
  "msg": "Successfully released 5 tokens from Operation Id opr_121 to
Account Id:
oaccount~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376152
df (org_id : Org1MSP, user_id : user1)",
}
```

**getOnHoldIds**
This method returns a list of all of the holding IDs for a specified account.

```
Ctx.Account.getOnHoldIds(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with a JSON object that lists all holding IDs for the specified account. On error, a rejection with an error message.

Return Value Example:

```
{
   "msg":"Holding Ids are: ohold~digicur~digiCurr101~opr_121",
   "holding_ids":[
      "ohold~digicur~digiCurr101~opr_121"
   ]
}
```

**getOnHoldDetailsWithOperationId**

This method returns the on-hold transaction details for a specified operation ID and token.

```
Ctx.Hold.getOnHoldDetailsWithOperationId(token_id: string, operation_id:
string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a hold object that includes the following properties:
  - `holding_id` – The holding ID of the transaction.
  - `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
  - `from_account_id` – The account ID of the current owner of the on-hold tokens.
  - `to_account_id` – The account ID of the receiver.
  - `notary_account_id` – The account ID of the notary.
  - `token_id: string` – The ID of the saved token.
  - `quantity` – The amount of tokens that are on hold for the holding ID.
  - `time_to_expiration` – The duration until the hold expires.
- On error, a rejection with an error message.

Return Value Example:

```
{
    "assetType": "ohold",
    "holding_id": "ohold~digicur~digiCurr101~opr_121",
    "operation_id": "opr_121",
    "token_name": "digicur",
    "from_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df",
    "to_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
    "notary_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
c706e",
    "token_id": "digiCurr101",
    "quantity": 10,
    "time_to_expiration": "2022-08-01T18:30:00.000Z"
}
```

**`getOnHoldBalanceWithOperationId`**
This method returns the on-hold balance for a specified operation ID and token. This method can be invoked by anyone.

```
Ctx.Hold.getOnHoldBalanceWithOperationId(token_id: string,
operation_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a promise object with the on-hold balance for the specified operation ID and token. On error, a rejection with an error message

Return Value Example:

```
{
    "msg": "Current Holding Balance of Operation 'op1' for token
'token1' is: 10",
    "holding_balance": 10
}
```

### Methods for Token Behavior Management - Burnable Behavior

**`burn`**
This method deactivates, or burns, tokens from the transaction caller's account. The caller of this method must have an account and the burner role. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.

```
Ctx.Token.burn(quantity: number, token: <Instance of Token Class>)
```

Parameters:

- `quantity: number` – The total number of tokens to burn.
- `token: <Instance of Token Class>` – The token asset to burn.

Returns:

- On success, a promise with a success message. On error, a rejection with an error message.

Return Value Example:

```
{
 "msg":"Successfully burned 10 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
```

```
f0376152df (Org-Id: Org1MSP, User-Id: admin)"
}
```

## TypeScript Methods for Token Conversion

Blockchain App Builder automatically generates methods that you can use to convert fungible tokens that use the Token Taxonomy Framework standard.

The token conversion methods include the concept of the **exchange pool**. The exchange pool account is funded by other token accounts. When you mint tokens, you can specify that a percentage of the minted tokens are transferred to the exchange pool account.

- Token Conversion Process
- Automatically Generated Token Conversion Methods
- Token Conversion SDK Methods

### Token Conversion Process

A typical flow for converting tokens follows these steps:

1. Call the `initializeExchangePoolUser` method to initialize the exchange pool user.
2. Call the `createExchangePoolAccounts` method to create exchange pool accounts. Create an exchange pool account for every type of fungible token that you want to convert from or convert to.
3. Call the `addConversionRate` method to set the conversion rate for each pair of tokens that you want to convert between.
4. Fund the exchange pool token accounts in one of the following ways:
   - Transfer tokens to the exchange pool token accounts using the standard transfer methods.
   - Call the `mintWithFundingExchangePoolToken` method when minting tokens, which can transfer a percentage of minted tokens to an exchange pool account.
5. Call the `tokenConversion` method to convert between two fungible tokens. A single user can convert tokens between two of their token accounts, or a pair of users can directly convert tokens from one account to another.
6. The exchange pool user can view the exchange pool account balances and account transactions.
   - Call the `getAccount` method to view the balances of each of the exchange pool token accounts.
   - Call the `getAccountTransactionHistory` and `getAccountTransactionHistoryWithFilters` methods to view account transactions for each of the exchange pool token accounts.

### Automatically Generated Token Conversion Methods

Blockchain App Builder automatically generates methods to convert between different types of fungible tokens. Controller methods must have a `@Validator(...params)` decorator to be invokable.

**initializeExchangePoolUser**

This method initializes the exchange pool user, which is a one-time activity. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async initializeExchangePoolUser(org_id: string, user_id:
string){
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.initializeExchangePo
olUser', 'TOKEN');
    return await
this.Ctx.TokenConvertor.initializeExchangePoolUser(org_id, user_id);
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the exchange pool user.

Return Value Example:

```
{
    "assetType": "oconversion",
    "convertor_id":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "org_id": "Org1MSP",
    "user_id": "exchangepooluser"
}
```

**createExchangePoolAccounts**

This method creates exchange pool token accounts for a given array of token IDs. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.array().of(yup.string()))
public async createExchangePoolAccounts(token_ids: string[]){
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.initializeExchangePo
olUser', 'TOKEN');
    return await
this.Ctx.TokenConvertor.createExchangePoolAccounts(token_ids);
}
```

Parameters:

- `token_ids: string []` – An array of token IDs.

Returns:

- On success, a list of objects that includes details of the exchange pool accounts that were created.

Return Value Example:

```
[
  {
    "account_id":
"oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f034",
    "token_id": "USD",
    "status": "created"
  },
  {
    "account_id":
"oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0",
    "token_id": "INR",
    "status": "created"
  }
]
```

**addConversionRate**

This method adds a conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.number())
public async addConversionRate(from_token_id:string , to_token_id:string,
token_conversion_rate: number) {
   await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.addConversionRate',
'TOKEN');
   return await
this.Ctx.TokenConvertor.addConversionToken(from_token_id,to_token_id,token_co
nversion_rate);
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: number` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
```

```
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491da
cf8d57873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 10
}
```

**getConversionRate**

This method gets the current conversion rate for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
@Validator(yup.string(), yup.string())
public async getConversionRate(from_token_id:string ,
to_token_id:string) {
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.getConversionRate',
'TOKEN');
    const conversion_rate_id = await
this.Ctx.TokenConversionRate.getConversionRateId(from_token_id,
to_token_id);
    return await this.Ctx.TokenConversionRate.get(conversion_rate_id);
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491da
cf8d57873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 10
}
```

**updateConversionRate**

This method updates the current conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.number())
public async updateConversionRate(from_token_id:string , to_token_id:string,
token_conversion_rate: number) {
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.updateConversionRate',
'TOKEN');
    return await
this.Ctx.TokenConvertor.updateTokenConversionRate(from_token_id,to_token_id,t
oken_conversion_rate);
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: number` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the updated conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 20
}
```

**mintWithFundingExchangePool**

This method mints tokens in the caller's account based on the specified token ID and quantity. A percentage of tokens from the minted quantity is then transferred to the exchange pool token account.

```
@Validator(yup.string(), yup.number(), yup.number())
public async mintWithFundingExchangePool(token_id: string, token_quantity:
number, percentage_token_to_exchange_pool: number) {
    return await
this.Ctx.TokenConvertor.mintWithFundingExchangePool(token_id,
```

```
token_quantity, percentage_token_to_exchange_pool);
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `token_quantity: number` – The total number of tokens to mint.
- `percentage_token_to_exchange_pool: number` – The percentage of minted tokens to transfer to the exchange pool token account.

Returns:

- On success, a message that indicates that minting and funding the exchange pool were successful.

Return Value Example:

```
{
    "msg": "Successfully minted 100 tokens to Account Id:
oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48
eb (Org-Id: Org1MSP, User-Id: admin) and Successfully transfered 20
tokens to exchange pool Account with Account Id:
oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f0
34 (Org-Id: Org1MSP, User-Id: exchangepooluser) "
}
```

**`tokenConversion`**

This method converts tokens from the caller's account to the account specified by the `to_token_id`, `to_org_id` and `to_user_id` values. This method can be called by the `Token Admin` of the chaincode and by any token account owner. An exchange pool user cannot call this method.

```
@Validator(yup.string(),yup.string(),yup.string(),yup.string(),yup.numb
er())
public async tokenConversion(from_token_id:string, to_token_id:string,
to_org_id:string, to_user_id:string, token_quantity:number){
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.tokenConversion',
'TOKEN');
    return await
this.Ctx.TokenConvertor.tokenConversion(from_token_id,to_token_id,to_or
g_id,to_user_id,token_quantity);
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `to_org_id: string` – The membership service provider (MSP) ID of the user in the current organization to receive the tokens.

- `to_user_id: string` – The user name or email ID of the user to receive the tokens.
- `token_quantity: number` – The total number of tokens to transfer.

Returns:

- On success, a message that indicates the token conversion was successful.

Return Value Example:

```
{
    "msg": "Succesfully converted 5 of tokens with tokenId: [USD] from
AccountId:
oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb
(Org-Id: Org1MSP, User-Id: admin) to 100 of tokens with tokenId: [INR] to
AccountId:
oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628f1e
(Org-Id: Org1MSP, User-Id: user) as per the conversion rate of 20"
}
```

**`getConversionHistory`**

This method returns the token conversion history for a specified token account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the token account owner.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async getConversionHistory(token_id: string, org_id: string,
user_id: string) {
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT.getConversionHistory",
"TOKEN", { account_id });
    return await this.Ctx.Account.getTokenConversionHistory(account_id,
org_id, user_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object with conversion history details.

Return Value Example:

```
[
  {
    "transaction_id":
```

```
"otransaction~34edd19e03ec8bbbc77bc3372081410a824a5c10f9aa522b3a6390d7e
8cb11cf",
    "from_account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",
    "to_account_id":
"oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628
f1e",
    "transacted_amount": 5,
    "converted_amount": 100,
    "conversion_rate": "20",
    "from_token_id": "USD",
    "to_token_id": "INR",
    "balance": 75,
    "onhold_balance": 0,
    "timestamp": "2022-11-30T11:03:20.000Z",
    "transaction_type": "TOKEN_CONVERSION_DEBIT"
  }
]
```

**`getConversionRateHistory`**

This method returns the token conversion rate history for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
@Validator(yup.string(), yup.string())
public async getConversionRateHistory(from_token_id:string ,
to_token_id:string) {
    const conversion_rate_id = await
this.Ctx.TokenConversionRate.getConversionRateId(from_token_id,to_token
_id);
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.getConversionRateHis
tory', 'TOKEN');
    return await
this.Ctx.TokenConversionRate.history(conversion_rate_id);
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from, for the purpose of calculating the conversion rate.
- `to_token_id: string` – The ID of the token to convert to, for the purpose of calculating the conversion rate.

Returns:

- On success, a JSON object with conversion rate history details.

Return Value Example:

```
 [
  {
    "trxId":
"0b1ba7bc2620e1438b6580365e5c0ab852247ccfa5a3eb2157d3baca02c0e521",
    "timeStamp": "2022-11-30T10:23:38.000Z",
    "value": {
      "assetType": "oconversionRate",
      "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
873b",
      "from_token_id": "USD",
      "to_token_id": "INR",
      "conversion_rate": 20
    }
  },
  {
    "trxId":
"36fc40ddb3d8308ee7e156af700da131d78d941fe390fc57985b7589e7035d5c",
    "timeStamp": "2022-11-30T10:13:18.000Z",
    "value": {
      "assetType": "oconversionRate",
      "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
873b",
      "from_token_id": "USD",
      "to_token_id": "INR",
      "conversion_rate": 10
    }
  }
]
```

**`getExchangePoolUser`**
This method returns the `org_id` and `user_id` values for the exchange pool user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator()
public async getExchangePoolUser() {
    await
this.Ctx.Auth.checkAuthorization('TOKEN_CONVERSION.getExchangePoolUser',
'TOKEN');
    return await this.Ctx.TokenConvertor.getExchangePoolUser();
}
```

Parameters:

- none

Returns:

- On success, a message with information about the exchange pool user.

Return Value Example:

```
{
   "assetType": "oconversion",
   "convertor_id":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
   "org_id": "Org1MSP",
   "user_id": "exchangepooluser"
}
```

### Token Conversion SDK Methods

**`initializeExchangePoolUser`**
This method initializes the exchange pool user, which is a one-time activity. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.initializeExchangePoolUser(orgId: string, userId:
string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the exchange pool user.

Return Value Example:

```
{
    "assetType": "oconversion",
    "convertor_id":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "org_id": "Org1MSP",
    "user_id": "exchangepooluser"
}
```

**`createExchangePoolAccounts`**
This method creates exchange pool token accounts for a given array of token IDs. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.createExchangePoolAccounts(token_ids: string[])
```

Parameters:

- `token_ids: string []` – An array of token IDs.

Returns:

- On success, a list of objects that includes details of the exchange pool accounts that were created.

Return Value Example:

```
[
  {
    "account_id":
"oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f034",
    "token_id": "USD",
    "status": "created"
  },
  {
    "account_id":
"oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0",
    "token_id": "INR",
    "status": "created"
  }
]
```

**addConversionToken**

This method adds tokens with a new conversion rate for a specified token. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.addConversionToken(fromTokenId: string, toTokenId:
string, tokenConversionRate: number)
```

Parameters:

- `fromTokenId: string` – The ID of the token to convert from.
- `toTokenId: string` – The ID of the token to convert to.
- `tokenConversionRate: number` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 10
}
```

**get**
This method gets the current conversion rate for a pair of tokens. This method can be called by the `Token Admin` of the chaincode and by any token account owner.

```
Ctx.TokenConversionRate.get(id: string)
```

Parameters:

- `id: string` – The ID of the token conversion rate object.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491da
cf8d57873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 10
}
```

**updateTokenConversionRate**
This method updates the current conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.updateTokenConversionRate(fromTokenId: string,
toTokenId: string, tokenConversionRate: number)
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: number` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the updated conversion rate object.

Return Value Example:

```
{
  "assetType": "oconversionRate",
  "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491da
```

```
cf8d57873b",
  "from_token_id": "USD",
  "to_token_id": "INR",
  "conversion_rate": 20
}
```

**mintWithFundingExchangePool**

This method mints tokens in the caller's account based on the specified token ID and quantity. A percentage of tokens from the minted quantity is then transferred to the exchange pool token account.

```
Ctx.TokenConvertor.mintWithFundingExchangePool(tokenId: string,
tokenQuantity: number, percentageTokenToExchangePool: number)
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `token_quantity: number` – The total number of tokens to mint.
- `percentage_token_to_exchange_pool: number` – The percentage of minted tokens to transfer to the exchange pool token account.

Returns:

- On success, a message that indicates that minting and funding the exchange pool were successful.

Return Value Example:

```
{
    "msg": "Successfully minted 100 tokens to Account Id:
oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb
(Org-Id: Org1MSP, User-Id: admin) and Successfully transfered 20 tokens to
exchange pool Account with Account Id:
oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f034
(Org-Id: Org1MSP, User-Id: exchangepooluser) "
}
```

**tokenConversion**

This method converts tokens from the caller's account to the account specified by the `to_token_id`, `to_org_id`, and `to_user_id` values. This method can be called by the `Token Admin` of the chaincode and by any token account owner. An exchange pool user cannot call this method.

```
Ctx.TokenConvertor.tokenConversion(fromTokenId: string, toTokenId: string,
toOrgId: string, toUserId: string, tokenQuantity: number)
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.

- `to_org_id: string` – The membership service provider (MSP) ID of the user in the current organization to receive the tokens.
- `to_user_id: string` – The user name or email ID of the user to receive the tokens.

Returns:

- On success, a message that indicates the token conversion was successful.

Return Value Example:

```
{
    "msg": "Succesfully converted 5 of tokens with tokenId: [USD] from
AccountId:
oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48
eb (Org-Id: Org1MSP, User-Id: admin) to 100 of tokens with tokenId:
[INR] to AccountId:
oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628f
1e (Org-Id: Org1MSP, User-Id: user) as per the conversion rate of 20"
}
```

**`getTokenConversionHistory`**
This method returns the token conversion history for a specified token account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, and by the token account owner.

```
Ctx.Account.getTokenConversionHistory(account_id: string, org_id:
string, user_id: string)
```

Parameters:

- `account_id: string` – The ID of the fungible token account.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object with conversion history details.

Return Value Example:

```
[
  {
    "transaction_id":
"otransaction~34edd19e03ec8bbbc77bc3372081410a824a5c10f9aa522b3a6390d7e
8cb11cf",
    "from_account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",
    "to_account_id":
```

```
"oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628f1e",
    "transacted_amount": 5,
    "converted_amount": 100,
    "conversion_rate": "20",
    "from_token_id": "USD",
    "to_token_id": "INR",
    "balance": 75,
    "onhold_balance": 0,
    "timestamp": "2022-11-30T11:03:20.000Z",
    "transaction_type": "TOKEN_CONVERSION_DEBIT"
  }
]
```


**history**

This method returns the token conversion rate history for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
Ctx.TokenConversionRate.history(conversion_rate_id: string)
```

Parameters:

- `conversion_rate_id: string` – The ID of the conversion rate object.

Returns:

- On success, a JSON object with conversion rate history details.

Return Value Example:


```
 [
  {
    "trxId":
"0b1ba7bc2620e1438b6580365e5c0ab852247ccfa5a3eb2157d3baca02c0e521",
    "timeStamp": "2022-11-30T10:23:38.000Z",
    "value": {
      "assetType": "oconversionRate",
      "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
873b",
      "from_token_id": "USD",
      "to_token_id": "INR",
      "conversion_rate": 20
    }
  },
  {
    "trxId":
"36fc40ddb3d8308ee7e156af700da131d78d941fe390fc57985b7589e7035d5c",
    "timeStamp": "2022-11-30T10:13:18.000Z",
    "value": {
      "assetType": "oconversionRate",
      "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d57
```

```
873b",
      "from_token_id": "USD",
      "to_token_id": "INR",
      "conversion_rate": 10
    }
  }
]
```

**getExchangePoolUser**
This method returns the `OrgId` and `UserId` values for the exchange pool user. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.getExchangePoolUser()
```

Parameters:

- none

Returns:

- On success, a message with information about the exchange pool user.

Return Value Example:

```
{
   "assetType": "oconversion",
   "convertor_id":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
   "org_id": "Org1MSP",
   "user_id": "exchangepooluser"
}
```

## TypeScript Methods for Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for fungible tokens that use the Token Taxonomy Framework standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, transferring, and holding tokens. Additionally, other users cannot transfer tokens to or hold tokens in a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.

- Automatically Generated Account Status Methods
- Account Status SDK Methods

### Automatically Generated Account Status Methods

Blockchain App Builder automatically generates methods to manage token account status. Controller methods must have a `@Validator(...params)` decorator to be invokable.

**`getAccountStatus`**
This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the token account owner. This method also supports data migration for existing chaincode that is upgraded to a newer version.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async getAccountStatus(token_id: string, org_id: string, user_id:
string) {
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT_STATUS.get", "TOKEN",
{ account_id });
    try {
      return await this.Ctx.AccountStatus.getAccountStatus(account_id);
    } catch (err) {
      return await
this.Ctx.AccountStatus.getDefaultAccountStatus(account_id);
    }
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the token account status.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

**getAccountStatusHistory**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the token account owner.

```
  @Validator(yup.string(), yup.string(), yup.string())
  public async getAccountStatusHistory(token_id: string, org_id:
string, user_id: string) {
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    await this.Ctx.Account.getAccount(account_id);
    await this.Ctx.Auth.checkAuthorization("ACCOUNT_STATUS.history",
"TOKEN", { account_id });
    const status_id = await
this.Ctx.AccountStatus.generateAccountStatusId(account_id);
    let account_status_history: any;
    try {
      account_status_history = await
this.Ctx.AccountStatus.history(status_id);
    } catch (err) {
      return [];
    }
    return account_status_history;
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the account status history.

Return Value Example:

```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "suspended"
```

```
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "active"
    }
  }
]
```

**activateAccount**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization. Deleted accounts cannot be activated.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async activateAccount(token_id: string, org_id: string, user_id:
string) {
    await this.Ctx.Auth.checkAuthorization("ACCOUNT_STATUS.activateAccount",
"TOKEN", { org_id });
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    return await this.Ctx.Account.activateAccount(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
```

```
6d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
  "status": "active"
}
```

**suspendAccount**

This method suspends a token account. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async suspendAccount(token_id: string, org_id: string,
user_id: string) {
    await
this.Ctx.Auth.checkAuthorization("ACCOUNT_STATUS.suspendAccount",
"TOKEN", { org_id });
    const account_id = await
this.Ctx.Account.generateAccountId(token_id, org_id, user_id);
    return await this.Ctx.Account.suspendAccount(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "status": "suspended"
}
```

**deleteAccount**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization. After an

account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot be changed to any other state. To delete an account, the account balance and the on-hold balance must be zero.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async deleteAccount(token_id: string, org_id: string, user_id:
string) {
    await this.Ctx.Auth.checkAuthorization("ACCOUNT_STATUS.deleteAccount",
"TOKEN", { org_id });
    const account_id = await this.Ctx.Account.generateAccountId(token_id,
org_id, user_id);
    return await this.Ctx.Account.deleteAccount(account_id);
  }
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

### Account Status SDK Methods

**`getAccountStatus`**
This method gets the current status of the token account.

```
Ctx.AccountStatus.getAccountStatus(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "status": "active"
}
```

**saveAccountStatus**
This method saves the status object (if a status object is not present) for the token account, and sets the status to the specified value.

```
Ctx.AccountStatus.saveAccountStatus(account_id: string, status:
AccountStatus)
```

Parameters:

- `account_id: string` – The ID of the token account.
- `status: AccountStatus` – The status to set for the specified account.
  `AccountStatus` is an `enum` type which must be `active`, `suspended`, or `deleted`.

Returns:

- On success, a JSON representation of the account status object.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "status": "active"
}
```

**getAccountStatusHistory**
This method gets the history of the account status.

```
Ctx.AccountStatus.history(status_id: string)
```

Parameters:

- `status_id: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:

```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "suspended"
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "active"
    }
  }
]
```

**`activateAccount`**
This method activates a token account.

```
Ctx.Account.activateAccount(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
  "status": "active"
}
```

**suspendAccount**
This method suspends a token account.

```
Ctx.Account.suspendAccount(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "status": "suspended"
}
```

**deleteAccount**
This method deletes a token account.

```
Ctx.Account.deleteAccount(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

## Scaffolded Go Project for Token Taxonomy Framework

Blockchain App Builder takes the input from your token specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates token lifecycle classes and functions, including CRUD and non-CRUD methods. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to tokens, see Scaffolded Go Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated Token Methods
  - Custom Methods
- Token SDK Methods

### Model

Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class.

```
package src
type Digicur struct {
    AssetType               string                  `json:"AssetType"
final:"otoken"`
    Token_id                string                  `json:"Token_id"
id:"true" mandatory:"true" validate:"regexp=^[A-Za-z0-9][A-Za-
z0-9_-]*$,max=16"`
    Token_name              string                  `json:"Token_name"
final:"digicur"`
    Token_desc              string                  `json:"Token_desc"
validate:"max=256"`
    Token_type              string                  `json:"Token_type"
final:"fungible" validate:"regexp=^fungible$"`
    Behavior                []string                `json:"Behavior"
final:"[\"divisible\",\"mintable\",\"transferable\",\"burnable\",\"holdable\"
,\"roles\"]"`
    Roles                   map[string]interface{} `json:"Roles"
final:"{\"minter_role_name\":\"minter\",\"burner_role_name\":\"burner\",\"not
```

```
ary_role_name\":\"notary\"}"`
    Mintable                map[string]interface{} `json:"Mintable"
final:"{\"Max_mint_quantity\":20000}"`
    Divisible               map[string]interface{} `json:"Divisible"
final:"{\"Decimal\":1}"`
    Token_to_currency_ratio int
`json:"Token_to_currency_ratio" validate:"int"`
    Currency_representation string
`json:"Currency_representation" validate:"string"`
    Metadata                interface{}
`json:"Metadata,omitempty"`
}
```

### Controller

There is only one main controller.

```
type Controller struct {
    Ctx trxcontext.TrxContext
}
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

If you use more than one token SDK method in a custom method, do not use methods that will affect the same key-value pairs in the state database.

Instead, use the `BulkTransferTokens` method to transfer to multiple accounts from the caller's account, as shown in the following code snippet.

```
BulkTransferTokens(token_id string, flow: []map[string]interface{})
```

> **Note:**
>
> If you use more than one token SDK method in a custom method that might affect the same key-value pairs in the state database, enable the MVCC optimization for token chaincodes. For more information, see MVCC Optimization.

### Automatically Generated Token Methods

Blockchain App Builder automatically generates methods to support tokens and token life cycles. You can use these methods to initialize tokens, manage roles and accounts, and complete other token lifecycle tasks without any additional coding. Controller methods must be public to be invokable. Public method names begin with an upper case character. Method names that begin with a lower case character are private.

- Access Control Management

- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Holdable Behavior
  - Burnable Behavior

**Methods for Access Control Management**

**`AddTokenAdmin`**

This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenAdmin(org_id string, user_id string)
(interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Admin.AddAdmin", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Admin.AddTokenAdmin(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{
    "msg": "Successfully added Token Admin (Org_Id: Org1MSP, User_Id: user1)"
}
```

**RemoveTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) RemoveTokenAdmin(org_id string, user_id string)
(interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Admin.RemoveAdmin",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.Admin.RemoveAdmin(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfuly removed Admin (Org_Id Org1MSP User_Id user1)"}
```

**IsTokenAdmin**

This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`. A `Token Admin` or `Org Admin` can call this function on any other user in the blockchain network. Other users can call this method only on their own accounts.

```
func (t *Controller) IsTokenAdmin(org_id string, user_id string)
(interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Admin.IsTokenAdmin",
"TOKEN", map[string]string{"org_id": org_id, "user_id": user_id})
      if err != nil || !auth {
            return false, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Auth.IsUserTokenAdmin(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.

- `user_id string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

Return Value Example:

```
{"result":false}
```

**GetAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by the `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetAllTokenAdmins() (interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Admin.GetAllAdmins", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.Admin.GetAllAdmins()
}
```

Parameters:

- none

Returns:

- On success, a JSON list of admins that includes `OrgId` and `UserId` objects.

Return Value Example:

```
{"admins":[{"OrgId":"Org1MSP","UserId":"admin"},
{"OrgId":"Org1MSP","UserId":"user2"}]}
```

**AddOrgAdmin**

This method adds a user as an `Org Admin` of the organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
func (t *Controller) AddOrgAdmin(org_id string, user_id string)
(interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Admin.AddOrgAdmin",
"TOKEN", map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Admin.AddOrgAdmin(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully added Org Admin (Org_Id: Org1MSP, User_Id:
orgAdmin)"
}
```

**`RemoveOrgAdmin`**

This method removes a user as an `Org Admin` of an organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
func (t *Controller) RemoveOrgAdmin(org_id string, user_id string)
(interface{}, error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Admin.RemoveOrgAdmin", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Admin.RemoveOrgAdmin(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully removed Org Admin (Org_Id Org1MSP User_Id
orgAdmin)"
}
```

**GetOrgAdmins**

This method returns a list of all users who are an `Org Admin` of an organization. This method can be called only by a `Token Admin` of the chaincode or by any `Org Admin`.

```
func (t *Controller) GetOrgAdmins() (interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Admin.GetOrgAdmins",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Admin.GetAllOrgAdmins()
}
```

Parameters:

- none

Returns:

- On success, a JSON list that includes `OrgId` and `UserId` objects.

Return Value Example:

```
{
    "admins": [
        {
            "OrgId": "Org1MSP",
            "UserId": "orgadmin"
        },
        {
            "OrgId": "Org1MSP",
            "UserId": "orgadmin1"
        },
        {
            "OrgId": "Org1MSP",
            "UserId": "orgadmin2"
        }
    ]
}
```

### Methods for Token Configuration Management

**Init**

This method is called when the chaincode is deployed. Every `Token Admin` is identified by the `user_id` and `org_id` information in the mandatory `adminList` parameter. The `user_id` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `org_id` is the membership service provider (MSP) ID of the user in the current network organization.

Any `Token Admin` user can add and remove other `Token Admin` users by calling the `AddTokenAdmin` and `RemoveTokenAdmin` methods.

```
func (t *Controller) Init(adminList []admin.TokenAdminAsset)
(interface{}, error) {
    list, err := t.Ctx.Admin.InitAdmin(adminList)
    if err != nil {
        return nil, fmt.Errorf("initializing admin list failed %s",
err.Error())
    }
    return list, nil
}
```

Parameters:

- `adminList array` – An array of `{user_id, org_id}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

Parameter example, Mac OSX and Linux CLI:

```
'[{"user_id":"userid", "org_id":"OrgMSPId"}]'
```

Parameter example, Microsoft Windows CLI:

```
"[{\"user_id\":\"userid\", \"org_id\":\"OrgMSPId\"}]"
```

Parameter example, Oracle Blockchain Platform console:

```
["[{\"user_id\":\"userid\", \"org_id\":\"OrgMSPId\"}]"]
```

**`Initialize<Token Name>Token`**
This method creates a token and initializes the token properties. The asset and its properties are saved in the state database. This method can be invoked only by a `Token Admin` of the chaincode.

```
func (t *Controller) InitializeDigicurToken(asset Digicur)
(interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.Save", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    return t.Ctx.Token.Save(&asset)
}
```

Parameters:

- `asset <Token Class>` – The token asset is passed as the parameter to this method. The properties of the token asset are described in the model file.

Returns:

- On success, a JSON representation of the token asset that was created.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "",
    "Token_type": "fungible",
    "Behavior": ["divisible", "mintable", "transferable", "burnable",
"roles"],
    "Roles": {
        "minter_role_name": "minter"
    },
    "Mintable": {
        "Max_mint_quantity": 1000
    },
    "Divisible": {
        "Decimal": 2
    },
    "Currency_name": "",
    "Token_to_currency_ratio": 1
}
```

**Update<Token Name>Token**
This method updates token properties. After a token asset is created, only the `token_desc` property and custom properties can be updated. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) UpdateDigicurToken(asset Digicur) (interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.Update", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    return t.Ctx.Token.Update(&asset)
}
```

Parameters:

- `asset <Token Class>` – The token asset is passed as the parameter to this method. The properties of the token asset are described in the model file.

Returns:

- On success, an updated JSON representation of the token asset.

Return Value Example:

```
{
    "AssetType": "otoken",
```

```
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "Digital Currency equiv of dollar",
    "Token_type": "fungible",
    "Behavior": ["divisible", "mintable", "transferable", "burnable",
"roles"],
    "Roles": {
        "minter_role_name": "minter"
    },
    "Mintable": {
        "Max_mint_quantity": 1000
    },
    "Divisible": {
        "Decimal": 2
    },
    "Currency_name": "",
    "Token_to_currency_ratio": 1
}
```

**GetTokenDecimals**

This method returns the number of decimal places that were configured for a fractional token. If the `divisible` behavior was not specified for the token, then the default value is 0. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetTokenDecimals(token_id string) (interface{},
error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("Token.GetTokenDecimals", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    tokenDecimal, err := t.Ctx.Token.GetTokenDecimals(token_id)
    if err != nil {
        return nil, fmt.Errorf("Error in GetTokenDecimals %s",
err.Error())
    }
    response := make(map[string]interface{})
    response["msg"] = fmt.Sprintf("Token Id: %s has %d decimal
places.", token_id, tokenDecimal)
    return response, nil
}
```

Parameters:

- `token_id string` – The ID of the token.

Returns:

- On success, a JSON string showing the number of token decimal places.

Return Value Example:

```
{"msg":"Token Id: digiCurr101 has 1 decimal places."}
```

**GetTokenById**

This method returns a token object if it is present in the state database. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetTokenById(token_id string) (interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.Get", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    tokenAsset, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return tokenAsset.Interface(), err
}
```

Parameters:

- `token_id string` – The ID of the token.

Returns:

- On success, a JSON object that represents the token asset.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "Digital Currency equiv of dollar",
    "Token_type": "fungible",
    "Behavior": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "Roles": {
        "minter_role_name": "minter"
    },
    "Mintable": {
        "Max_mint_quantity": 1000
    },
    "Divisible": {
        "Decimal": 2
```

```
    },
    "Currency_name": "",
    "Token_to_currency_ratio": 1
}
```

**GetTokenHistory**

This method returns the token history for a specified token ID. Any user can call this method.

```
func (t *Controller) GetTokenHistory(token_id string) (interface{},
error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Token.GetTokenHistory", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Token.History(token_id)
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object that represents the token history.

Return Value Example:

```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-09-01T16:46:33Z",
        "TxId":
"12333b8a4f63aa9b3a34072efcbd7df546c6d1e7d82a7a9596e899383656d6f7",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "Currency_name1": "",
            "Divisible": {
                "Decimal": 2
            },
            "Mintable": {
                "Max_mint_quantity": 1000
            },
            "Roles": {
```

```
                "minter_role_name": "minter"
            },
            "Token_desc": "updated description",
            "Token_id": "token",
            "Token_name": "fiatmoneytok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-09-01T16:04:25Z",
        "TxId":
"99702e2dad7554a5ee4716a0d01d3e394cbce39bea8bade265d8911f30ebad0b",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "Currency_name1": "",
            "Divisible": {
                "Decimal": 2
            },
            "Mintable": {
                "Max_mint_quantity": 1000
            },
            "Roles": {
                "minter_role_name": "minter"
            },
            "Token_desc": "",
            "Token_id": "token",
            "Token_name": "fiatmoneytok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```


**GetAllTokens**

This method returns all tokens that are stored in the state database. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetAllTokens() (interface{}, error) {
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.GetAllTokens", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
```

```
err.Error())
    }
    return t.Ctx.Token.GetAllTokens()
}
```

Parameters:

- none

Returns:

- On success, a JSON object that represents all token assets.

Return Value Example:

```
"payload": [
    {
        "key": "t1",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "Currency_name value",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "Token_desc value",
            "Token_id": "t1",
            "Token_name": "obptok",
            "Token_to_currency_ratio": 2,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```

**GetTokensByName**

This method returns all token objects with a specified name. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode. This method uses

Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetTokensByName(token_name string) (interface{}, error)
{
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.GetTokensByName",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.Token.GetTokensByName(token_name)
}
```

Parameters:

- `token_name string` – The name of the tokens to retrieve. The name corresponds to the `Token_name` property in the specification file. The value is the class name of the token.

Returns:

- On success, a JSON object of all token assets that match the name.

Return Value Example:

```
"payload": [
    {
        "key": "t1",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "Currency_name value",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "Token_desc value",
            "Token_id": "t1",
            "Token_name": "obptok",
```

```
            "Token_to_currency_ratio": 999,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    },
    {
        "key": "obp2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "",
            "Token_id": "obp2",
            "Token_name": "obptok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```


### Methods for Account Management

**`CreateAccount`**

This method creates an account for a specified user and token. An account must be created for any user who will have tokens at any point. Accounts track balances, on-hold balances, and transation history. An account ID is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`org_id`) of the user in the

current network organization. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization.

```
unc (t *Controller) CreateAccount(org_id string, user_id string, token_type
string) (interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Account.CreateAccount",
"TOKEN", map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Account.CreateAccount(org_id, user_id, token_type)
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.
- `token_type: string` – The type of the token, which must be `fungible`.

Returns:

- On success, a JSON object of the account that was created. The `BapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
{
   "AssetType":"oaccount",

"AccountId":"oaccount~a73085a385bc96c4a45aa2dff032e7dede82c0664dee5f396b7c585
4eeafd4bd",
   "BapAccountVersion": 0,
   "UserId":"user1",
   "OrgId":"Org1MSP",
   "AccountType":"fungible",
   "TokenId":"",
   "TokenName":"",
   "Balance":0,
   "BalanceOnHold":0
}
```

**`AssociateTokenToAccount`**

This method associates a fungible token with an account. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the relevant organization.

```
func (t *Controller) AssociateTokenToAccount(account_id string, token_id
string) (interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Account.AssociateToken",
"TOKEN", map[string]string{"account_id": account_id})
```

```
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Account.AssociateToken(account_id, token_id)
}
```

Parameters:

- `account_id string` – The ID of the account.
- `token_id string` – The ID of the token.

Returns:

- On success, a JSON object of the updated account. The `BapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
{
"AssetType":"oaccount",
"AccountId":"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a
7eb54bdbebf48eb",
"BapAccountVersion": 0,
"UserId":"admin",
"OrgId":"Org1MSP",
"AccountType":"fungible",
"TokenId":"token1",
"TokenName":"loyaltok",
"Balance":0,
"BalanceOnHold":0
}
```

**`GetAccount`**

This method returns account details for a specified user and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
func (t *Controller) GetAccount(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("Account.GetAccount",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
```

```
      return t.Ctx.Account.GetAccountWithStatus(account_id)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `AccountId` – The ID of the user account.
- `UserId` – The user name or email ID of the user.
- `OrgId` – The membership service provider (MSP) ID of the user in the current organization.
- `TokenId` – The ID of the token.
- `Balance` – The current balance of the account.
- `BalanceOnHold` – The current on-hold balance of the account.
- `BapAccountVersion` – An account object parameter for internal use.
- `Status` – The current status of the user account.

Return Value Example:

```
{
  "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
  "AssetType": "oaccount",
  "Balance": 95,
  "BalanceOnHold": 0,
  "BapAccountVersion": 8,
  "OrgId": "appdev",
  "Status": "active",
  "TokenId": "obp1",
  "TokenName": "obptok",
  "TokenType": "fungible",
  "UserId": "idcqa"
}
```

**`GetAccountHistory`**

This method returns account history details for a specified user and token. This method can be called only by a `Token Admin` of the chaincode or the `AccountOwner` of the account.

```
func (t *Controller) GetAccountHistory(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
```

```
user_id)
      if err != nil {
            return nil, err
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("Account.History",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Account.History(account_id)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, an array of JSON account objects that includes the following properties:
- `TxId` – The transaction ID of the transaction as returned by the ledger.
- `Timestamp` – The time of the transaction.
- `IsDelete` – A Boolean value that indicates whether the record is deleted.
- `Value` – A JSON string of the account object. The `BapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
[
  {
      "IsDelete": "false",
      "Timestamp": "2023-08-28T19:31:15Z",
      "TxId":
"adde470a63860ec1013bd5c5987e8a506a48942a91b0f39fc8e561374042bd27",
      "Value": {
          "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
          "AssetType": "oaccount",
          "Balance": 100,
          "BalanceOnHold": 0,
          "BapAccountVersion": 1,
          "OrgId": "Org1MSP",
          "TokenId": "t1",
          "TokenName": "obptok",
          "TokenType": "fungible",
```

```
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-08-28T19:30:23Z",
        "TxId":
"8fbeda2ba60ba175091faae5ae369247775f2cba45c4d6d1ead6f0b05be84743",
        "Value": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "Org1MSP",
            "TokenId": "t1",
            "TokenName": "obptok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-08-28T19:29:54Z",
        "TxId":
"19bb296ae71709e91b097ba5d9ebd7f7522095880382fbf5913334a46a6026aa",
        "Value": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "Org1MSP",
            "TokenId": "",
            "TokenName": "",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```


**GetAccountOnHoldBalance**

This method returns the current on-hold balance for a specified account and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
func (t *Controller) GetAccountOnHoldBalance(token_id string, org_id string,
user_id string) (interface{}, error) {
    account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
```

```
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAccountOnHoldBalance",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      response, err :=
t.Ctx.Account.GetAccountOnHoldBalance(account_id)
      return response, err
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the current on-hold balance.

Return Value Example:

```
{
    "holding_balance": 0,
    "msg": "Total Holding Balance of Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id: Org1MSP, user_id: user1) is 0"
}
```

**`GetAllAccounts`**

This method returns a list of all accounts. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAllAccounts() (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAllAccounts", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.Account.GetAllAccounts()
}
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
        "valueJson": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 100,
            "BalanceOnHold": 0,
            "BapAccountVersion": 1,
            "OrgId": "appdev",
            "TokenId": "t1",
            "TokenName": "obptok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```

**GetUserByAccountId**
This method returns user details (`org_id` and `user_id`) for a specified account. This method can be called by any user of the chaincode.

```
func (t *Controller) GetUserByAccountId(account_id string) (interface{},
error) {
    return t.Ctx.Account.GetUserByAccountById(account_id)
}
```

Parameters:

- `account_id string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`org_id`, `token_id`, and `user_id`).

Return Value Example:

```
{"org_id":"Org1MSP","token_id":"digiCurr101","user_id":"user1"}
```

**GetAccountBalance**

This method returns the current balance for a specified account and token. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
func (t *Controller) GetAccountBalance(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAccountBalance", "TOKEN",
map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      response, err := t.Ctx.Account.GetAccountBalance(account_id)
      return response, err
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the current account balance.

Return Value Example:

```
{"msg":"Current Balance of
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f is 0","user_balance":0}
```

**GetAllOrgAccounts**

This method returns a list of all token accounts that belong to a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
func (t *Controller) GetAllOrgAccounts(org_id string) (interface{},
error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAllOrgAccounts", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
```

```
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Account.GetAllOrgAccounts(org_id)
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts for the specified organization.

Return Value Example:

```
[
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
        "valueJson": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "appdev",
            "TokenId": "token",
            "TokenName": "fiatmoneytok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "key":
"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a850",
        "valueJson": {
            "AccountId":
"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a850",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "appdev",
            "TokenId": "token",
            "TokenName": "fiatmoneytok",
            "TokenType": "fungible",
            "UserId": "example_minter"
        }
    }
]
```

### Methods for Role Management

**AddRole**

This method adds a role to a specified user and token. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization who also has the specified role.

```
func (t *Controller) AddRole(token_id string, user_role string, org_id
string, user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      tokenAssetValue, err := t.getTokenObject(token_id)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Token.AddRoleMember", "TOKEN",
map[string]string{"org_id": org_id, "token_id": token_id, "role":
user_role})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Token.AddRoleMember(user_role, account_id,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `user_role string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
 {"msg":"Successfully added role minter to
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1)"}
```

**RemoveRole**

This method removes a role from a specified user and token. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization who also has the specified role.

```
func (t *Controller) RemoveRole(token_id string, user_role string, org_id
string, user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
      if err != nil {
            return nil, err
      }
      tokenAssetValue, err := t.getTokenObject(token_id)
      if err != nil {
            return nil, err
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("Token.RemoveRoleMember",
"TOKEN", map[string]string{"org_id": org_id, "token_id": token_id, "role":
user_role})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Token.RemoveRoleMember(user_role, account_id,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `user_role string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg":"successfully removed member_id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1) from role minter"}
```

**GetAccountsByRole**

This method returns a list of all account IDs for a specified role and token. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetAccountsByRole(token_id string, user_role
string) (interface{}, error) {
    auth, err:=
t.Ctx.Auth.CheckAuthorization("Role.GetAccountsByRole", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    return t.Ctx.Role.GetAccountsByRole(token_id, user_role)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `user_role string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{
    "accounts": [

"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4"
    ]
}
```

**GetAccountsByUser**

This method returns a list of all account IDs for a specified organization ID and user ID. This method can be called only by a `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the `Account Owner` specified in the parameters.

```
func (t *Controller) GetAccountsByUser(org_id string, user_id string)
(interface{}, error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAccountsByUser", "TOKEN",
map[string]string{"org_id": org_id, "user_id": user_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Account.GetAccountsByUser(org_id, user_id)
}
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{
    "accounts": [

"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4"
    ]
}
```

**GetUsersByRole**

This method returns a list of all users for a specified role and token. This method can be called only by a `Token Admin` of the chaincode or by the `Account Owner` specified in the parameters.

```
func (t *Controller) GetUsersByRole(token_id string, user_role string)
(interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Role.GetUsersByRole",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Role.GetUsersByRole(token_id, user_role)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `user_role string` – The name of the role to search for.

Returns:

- On success, a JSON array of the user objects (`org_id` and `user_id`).

Return Value Example:

```
{"Users":[{"org_id":"Org1MSP","token_id":"digiCurr101","user_id":"user1"}]}
```

**IsInRole**

This method returns a Boolean value to indicate if a user and token has a specified role. This method can be called only by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
func (t *Controller) IsInRole(token_id string, org_id string, user_id
string, user_role string) (interface{}, error) {
      tokenAssetValue, err := t.getTokenObject(token_id)
      if err != nil {
            return nil, err
      }
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("Token.IsInRole",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      result, err := t.Ctx.Token.IsInRole(user_role, account_id,
tokenAssetValue.Interface())
      if err != nil {
            return nil, fmt.Errorf("error in IsInRole %s", err.Error())
      }
      response := make(map[string]interface{})
      response["result"] = result
      return response, nil
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.
- `user_role string` – The name of the role to search for.

Returns:

- On success, a JSON string of the Boolean result.

Return Value Example:

```
{"result":false}
```

**GetOrgUsersByRole**

This method returns information about all users that have a specified role in a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
func (t *Controller) GetOrgUsersByRole(token_id string, user_role string,
org_id string) (interface{}, error) {
      auth, err := t.Ctx.Auth.CheckAuthorization("Role.GetOrgUsersByRole",
"TOKEN", map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Role.GetOrgUsersByRole(token_id, user_role, org_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all users with the specified role in the specified organization.

Return Value Example:

```
{
    "Users": [
        {
            "org_id": "Org1MSP",
            "token_id": "token",
            "user_id": "admin"
        },
        {
            "org_id": "Org1MSP",
            "token_id": "token",
            "user_id": "orgAdmin"
        }
    ]
}
```

**GetOrgAccountsByRole**

This method returns information about all accounts that have a specified role in a specified organization. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

```
func (t *Controller) GetOrgAccountsByRole(token_id string, user_role string,
org_id string) (interface{}, error) {
      auth, err :=
```

```
t.Ctx.Auth.CheckAuthorization("Role.GetOrgAccountsByRole", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Role.GetOrgAccountsByRole(token_id, user_role,
org_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts with the specified role in the specified organization.

Return Value Example:

```
{
    "accounts": [

"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",

"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a
850"
    ]
}
```

### Methods for Transaction History Management

**`GetAccountTransactionHistory`**

This method returns an array of account transaction history details for a specified user and token. This method can be called only by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account.

```
func (t *Controller) GetAccountTransactionHistory(token_id string,
org_id string, user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAccountTransactionHistory",
"TOKEN", map[string]string{"account_id": account_id})
```

```
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }

      transactionArray, err :=
t.Ctx.Account.GetAccountTransactionHistory(account_id, org_id, user_id)
      return transactionArray, err
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, an array of JSON account transaction objects that includes the following properties:
- `balance` – The account balance.
- `holding_id` – The ID of a holding account.
- `onhold_balance` – The on-hold balance.
- `timestamp` – The time of the transaction.
- `token_id` – The ID of the token.
- `transacted_account` – The account with which the transaction took place.
- `transacted_amount` – The amount of the transaction.
- `transaction_id` – The ID of the transaction.
- `transaction_type` – The type of transaction.

Return Value Example:

```
[{
    "balance": 199,
    "onhold_balance": 0,
    "timestamp": "2021-08-16T17:42:32.905+05:30",
    "token_id": "digiCurr101",
    "transacted_account":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
    "transacted_amount": 1,
    "transaction_id":
"otransaction~c8a9fa001aba6e0d8391b034655889df47eb5103713840b999a4ab41f5e57b3
8",
    "transaction_type": "DEBIT"
}, {
```

```
    "balance": 200,
    "onhold_balance": 0,
    "timestamp": "2021-08-16T17:41:59.262+05:30",
    "token_id": "digiCurr101",
    "transacted_account":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "transacted_amount": 100,
    "transaction_id":
"otransaction~65a0bf8ae8108baa7495fbab91c205651c055e9f480f6808753287173
026aa69",
    "transaction_type": "MINT"
}]
```

**GetAccountTransactionHistoryWithFilters**

This method returns an array of account transaction history details for a specified user and token. This method can be called only by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `AccountOwner` of the account. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAccountTransactionHistoryWithFilters(token_id
string, org_id string, user_id string,
filters ...account.AccountHistoryFilters) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetAccountTransactionHistoryWith
Filters", "TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }

      // sample format of filter: []string{"3", "",
"2022-01-16T15:16:36+00:00", "2022-01-17T15:16:36+00:00"}
      transactionArray, err :=
t.Ctx.Account.GetAccountTransactionHistoryWithFilters(account_id,
org_id, user_id, filters...)
      return transactionArray, err
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetAccountTransactionHistoryWithFilters 'token1' 'appbuilder12'
'user_minter'
'{"PageSize":10,"Bookmark":"1","StartTime":"2022-01-25T17:41:42Z","EndTime":"20
22-01-25T17:59:10Z"}'


[
  {
    "transaction_id":
"otransaction~3f9c306b0ef6994885939c1a6eb5f063b06617ecb932d4a043f323ba53d55f9
f",
    "transacted_amount": 200,
    "timestamp": "2022-02-15T18:27:13.000Z",
    "token_id": "token1",
    "transacted_account":
"oaccount~obptok~26e046c8ba8b98da2cdabb78113d67200581ea3d4eea5aa324
1abd3598e05d05",
    "transaction_type": "DEBIT",
    "balance": 9200,
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~f1d37c3abd5c85c0a399f246d8eb68257c49ab4fe4cdfd3501908583c51c421
e",
    "transacted_amount": 200,
    "timestamp": "2022-02-15T18:27:02.000Z",
    "token_id": "token1",
    "transaction_type": "BULKTRANSFER",
    "number_of_sub_transactions": 2,
    "balance": 9600,
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9
b",
    "transacted_amount": 200,
    "timestamp": "2022-02-15T18:26:57.000Z",
    "token_id": "token1",
    "transaction_type": "BULKTRANSFER",
    "number_of_sub_transactions": 2,
    "balance": 9800,
    "onhold_balance": 0
  },
  {
    "transaction_id":
```

```
"otransaction~07331a1f7be99d6750973674a783da9ec9ca17df23747cdf52d388865
d93f9a",
    "transacted_amount": 10000,
    "timestamp": "2022-02-15T18:26:30.000Z",
    "token_id": "token1",
    "transacted_account":
"oaccount~obptok~88b62f329f20fffc6fc9231cb51019a5e9550c78b657123d140897
62397d2b55",
    "transaction_type": "MINT",
    "balance": 10000,
    "onhold_balance": 0
  }
]
```

**GetSubTransactionsById**

This method returns an array of subtransaction history details for a specified transaction.

```
func (t *Controller) GetSubTransactionsById(transaction_id string)
(interface{}, error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetSubTransactionsById",
"TOKEN", map[string]string{"transaction_id": transaction_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Account.GetSubTransactionsById(transaction_id)
}
```

Parameters:

- `transaction_id string` – The ID of the transaction.

Example:

```
ochain invoke GetAccountSubTransactionHistory
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f8
64b9b'

[
  {
    "transaction_id":
"otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6
f864b9b~c4ca4238a0b923820dcc509a6f75849b",
    "transacted_amount": 100,
    "timestamp": "2022-02-15T18:26:57.000Z",
    "token_id": "token1",
    "transacted_account":
"oaccount~obptok~6600eb38d365552b76f41d4186acece104f31eae331a440f963e6f
a75b62ff21",
    "transaction_type": "DEBIT",
    "balance": 9900,
```

```
    "onhold_balance": 0
  },
  {
    "transaction_id":
"otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9
b~c81e728d9d4c2f636f067f89cc14862c",
    "transacted_amount": 100,
    "timestamp": "2022-02-15T18:26:57.000Z",
    "token_id": "token1",
    "transacted_account":
"oaccount~obptok~26e046c8ba8b98da2cdabb78113d67200581ea3d4eea5aa3241abd3598e0
5d05",
    "transaction_type": "DEBIT",
    "balance": 9800,
    "onhold_balance": 0
  }
]
```

**GetSubTransactionsByIdWithFilters**

This method returns an array of subtransaction history details for a specified transaction.

```
func (t *Controller) GetSubTransactionsByIdWithFilters(transaction_id
string, filters ...account.SubTransactionFilters) (interface{}, error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetSubTransactionsByIdWithFilters",
"TOKEN", map[string]string{"transaction_id": transaction_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Account.GetSubTransactionsByIdWithFilters(transaction_id,
filters...)
}
```

Parameters:

- `transaction_id string` – The ID of the transaction.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetAccountSubTransactionHistoryWithFilters
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9b'
'{"PageSize":10,"Bookmark":"1"}'

[
{
"transaction_id":
```

```
"otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6
f864b9b~c4ca4238a0b923820dcc509a6f75849b",
"transacted_amount": 100,
"timestamp": "2022-02-15T18:26:57.000Z",
"token_id": "token1",
"transacted_account":
"oaccount~obptok~6600eb38d365552b76f41d4186acece104f31eae331a440f963e6f
a75b62ff21",
"transaction_type": "DEBIT",
"balance": 9900,
"onhold_balance": 0
},
{
"transaction_id":
"otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6
f864b9b~c81e728d9d4c2f636f067f89cc14862c",
"transacted_amount": 100,
"timestamp": "2022-02-15T18:26:57.000Z",
"token_id": "token1",
"transacted_account":
"oaccount~obptok~26e046c8ba8b98da2cdabb78113d67200581ea3d4eea5aa3241abd
3598e05d05",
"transaction_type": "DEBIT",
"balance": 9800,
"onhold_balance": 0
}
]
```


**GetTransactionById**

This method returns the history of a `Transaction` asset.

```
func (t *Controller) GetTransactionById(transaction_id string)
(interface{}, error) {
    return t.Ctx.Transaction.GetTransactionById(transaction_id)
}
```

Parameters:

- `transaction_id string` – The ID of the transaction asset.

Returns:

- On success, an JSON array of the history for the transaction.

Return Value Example:


```
{
    "history": [
        {
            "IsDelete": "false",
            "Timestamp": "2021-08-16 20:19:05.028 +0530 IST",
            "TxId":
"67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb2220",
```

```
            "Value": {
                "Amount": 3,
                "AssetType": "otransaction",
                "FromAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
                "HoldingId": "ohold~digicur~digiCurr101~op2",
                "NumberOfSubTransactions": 0,
                "Timestamp": "2021-08-16T20:19:05+05:30",
                "ToAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
                "TokenId": "digiCurr101",
                "TransactionId":
"otransaction~67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb222
0",
                "TransactionType": "RELEASEHOLD"
            }
        }
    ],
    "transaction_id":
"otransaction~67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb222
0"
}
```

**DeleteHistoricalTransactions**

This method deletes older transactions from the state database.

```
func (t *Controller) DeleteHistoricalTransactions(timestamp string)
(interface{}, error) {
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Transaction.DeleteHistoricalTransactions",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Transaction.DeleteHistoricalTransactions(timestamp)
}
```

Parameters:

- `timestamp string` – A time stamp that indicates when to delete transactions. Transaction assets that are older than the specified time will be deleted.

### Methods for Token Behavior Management - Mintable Behavior

**IssueTokens**

This method mints tokens, which are then owned by the caller of the method. The caller must have an account and the minter role. The number of tokens that can be minted is limited by the `max_mint_quantity` property of `mintable` behavior in the specification file. If the `max_mint_quantity` property is not specified, an unlimited number of tokens can be minted. The quantity must be within the decimal values specified by the `decimal` parameter

of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account with the minter role.

```
func (t *Controller) IssueTokens(token_id string, quantity float64)
(interface{}, error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.Mint(quantity, tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `quantity float64` – The number of tokens to mint.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg":"Successfully minted 100 tokens to account
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1)"}
```

**GetTotalMintedTokens**

This method returns the total number of minted tokens for a specified token. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetTotalMintedTokens(token_id string)
(interface{}, error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.Auth.CheckAuthorization("Token.GetTotalMintedTokens", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    return
t.Ctx.Token.GetTotalMintedTokens(tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.

Returns:

- On success, a JSON string indicating the total number of tokens.

Return Value Example:

```
{"msg":"total minted amount for token with id digiCurr101 is
1000","quantity":1000}
```

**GetNetTokens**

This method returns the total net number of tokens available in the system for a specified token. The net token total is the amount of tokens remaining after tokens are burned. In equation form net tokens = total minted tokens - total burned tokens. If no tokens are burned, then the number of net tokens is equal to the total minted tokens. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) GetNetTokens(token_id string) (interface{}, error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    auth, err := t.Ctx.Auth.CheckAuthorization("Token.GetNetTokens", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    return t.Ctx.Token.GetNetTokens(tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.

Returns:

- On success, a JSON string indicating the net number of tokens.

Return Value Example:

```
{"msg":"net minted amount for token with id digiCurr101 is
1000","quantity":1000}
```

### Methods for Token Behavior Management - Transferable Behavior

**TransferTokens**

This method transfers tokens from the caller to a specified account. The caller of the method must have an account. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account.

```
func (t *Controller) TransferTokens(token_id string, to_org_id string,
to_user_id string, quantity float64) (interface{}, error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
```

```
        return nil, err
    }
    to_account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
to_org_id, to_user_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.Transfer(to_account_id, quantity,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `to_org_id string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `to_user_id string` – The user name or email ID of the receiver.
- `quantity float64` – The number of tokens to transfer.

Returns:

- On success, a message with details for both accounts.

Return Value Example:

```
{"msg":"successfully transferred 1 tokens from account
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1) to account
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (org_id : Org1MSP, user_id : admin)"}
```

**`BulkTransferTokens`**

This method is used to perform bulk transfer of tokens from the caller account to the accounts that are specified in the `flow` object. The quantities must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.The caller of this method must have an account already created. This method can be called only by the `AccountOwner` of the account.

```
func (t *Controller) BulkTransferTokens(token_id string,
flow[]map[string]interface{}) (interface{}, error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.BulkTransfer(flow, tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.

- `flow[]map[string]interface{}` – An array of JSON objects that specify receiver details and quantities.
  - `to_org_id string` – The membership service provider (MSP) ID of the receiver in the current organization.
  - `to_user_id string` – The user name or email ID of the receiver.
  - `quantity float64` – The number of tokens to transfer.

  For example:

```
[{
    "to_org_id": "Org1MSP",
    "to_user_id": "user1",
    "quantity": 10
}, {
    "to_org_id": "Org1MSP",
    "to_user_id": "user2",
    "quantity": 10
}]
```

Returns:

- A message indicating success.

Return Value Example:

```
{
    "from_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f",
    "msg": "Successfully transferred 2 tokens from Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (Org-Id: Org1MSP, User-Id: user1)",
    "sub_transactions": [
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
c706e"
        },
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df"
        }
    ]
}
```

**Methods for Token Behavior Management - Holdable Behavior**

**HoldTokens**

This method creates a hold on behalf of the owner of the tokens with the `to_account_id` account. A notary account is specified, which is responsible to either complete or release the hold. When the hold is created, the specified token balance from the payer is put on hold. A held balance cannot be transferred until the hold is either completed or released. The caller of this method must have an account already created. This method can be called only by the `AccountOwner` of the account.

```
func (t *Controller) HoldTokens(token_id string, operation_id string,
to_org_id string, to_user_id string, notary_org_id string,
notary_user_id string, quantity float64, TimeToExpiration string)
(interface{}, error) {
      tokenAssetValue, err := t.getTokenObject(token_id)
      if err != nil {
            return nil, err
      }
      notary_account_id, err :=
t.Ctx.Account.GenerateAccountId(token_id, notary_org_id,
notary_user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting notary account id
from org_id: %s and user_id: %s with token_id: %s, error %s ",
notary_org_id, notary_user_id, token_id, err.Error())
      }
      to_account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
to_org_id, to_user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting to_account id
from org_id: %s and user_id: %s with token_id: %s, error %s ",
to_org_id, to_user_id, token_id, err.Error())
      }
      return t.Ctx.Token.Hold(operation_id, to_account_id,
notary_account_id, quantity, TimeToExpiration,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `operation_id string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `to_org_id string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `to_user_id string` – The user name or email ID of the receiver.
- `notary_org_id string` – The membership service provider (MSP) ID of the notary in the current organization.
- `notary_user_id string` – The user name or email ID of the notary.
- `quantity float64` – The number of tokens to put on hold.

- `time_to_expiration` – The time when the hold expires. Specify `0` for a permanent hold. Otherwise use the RFC-3339 format. For example, `2021-06-02T12:46:06Z`.

Returns:

- On success, a message with the caller's account and hold details.

Return Value Example:

```
{"msg":"AccountId
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1) is successfully holding 2 tokens"}
```

**ExecuteHoldTokens**

This method completes a hold on a token. A quantity of tokens previously held by a token owner is transferred to a receiver. If the `quantity` value is less than the actual hold value, then the remaining amount is available again to the original owner of the tokens. This method can be called only by the `AccountOwner` ID with the `notary` role. The hold can only be completed by the notary.

```
func (t *Controller) ExecuteHoldTokens(token_id string, operation_id string,
quantity float64) (interface{} error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.ExecuteHold(operation_id, quantity,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `operation_id string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `quantity float64` – The number of on-hold tokens to transfer.

Returns:

- On success, a message with the caller's account ID and the quantity of the transaction.

Return Value Example:

```
{"msg":"Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1) has successfully executed '1'
tokens(digiCurr101) from the hold with Operation Id 'op1'"}
```

**ReleaseHoldTokens**

This method releases a hold on tokens. The transfer is not completed and all held tokens are available again to the original owner. This method can be called by the `Account Owner` ID

with the `notary` role within the specified time limit or by the payer, payee, or notary after the specified time limit.

```
func (t *Controller) ReleaseHoldTokens(token_id string, operation_id
string) (interface{} error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.ReleaseHold(operation_id,
tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `operation_id string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a message indicating that the hold was released.

Return Value Example:

```
{"msg":"Successfully released '3' tokens from Operation Id 'op2' to
Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1)"}
```

**GetOnHoldIds**

This method returns a list of all of the holding IDs for a specified account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or the `Account Owner` of the account.

```
func (t *Controller) GetOnHoldIds(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("Account.GetOnHoldIds", "TOKEN",
map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.Account.GetOnHoldIDs(account_id)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a JSON list of holding IDs. A holding ID is a concatenation of the `ohold` asset type, the name of the token, the token ID, and the operation ID.

Return Value Example:

```
{"holding_ids":["ohold~loyaltok123~t1~op1"],"msg":"Holding Ids are:
[ohold~loyaltok123~t1~op1]"}
```

**GetOnHoldDetailsWithOperationId**

This method returns the on-hold transaction details for a specified operation ID and token. This method can be invoked by anyone.

```
func (t *Controller) GetOnHoldDetailsWithOperationId(token_id string,
operation_id string) (interface{} error) {
    return t.Ctx.Hold.GetOnHoldDetailsWithOperationId(token_id, operation_id)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `operation_id string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a JSON hold object that includes the following properties:
- `HoldingId` – The holding ID of the transaction.
- `OperationId` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `FromAccountId` – The account ID of the current owner of the on-hold tokens.
- `ToAccountId` – The account ID of the receiver.
- `NotaryAccountId` – The account ID of the notary.
- `TokenId` – The ID of the saved token.
- `Quantity` – The amount of tokens that are on hold for the holding ID.
- `TimeToExpiration` – The duration until the hold expires.

Return Value Example:

```
{
    "AssetType": "ohold",
    "HoldingId": "ohold~digicur~digiCurr101~op1",
    "OperationId": "op1",
    "TokenName": "digicur",
    "FromAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "ToAccountId":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471a
aa1210c706e",
    "NotaryAccountId":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
    "TokenId": "digiCurr101",
    "Quantity": 2,
    "TimeToExpiration": "0"
}
```

**GetOnHoldBalanceWithOperationId**

This method returns the on-hold balance for a specified operation ID and token. This method can be invoked by anyone.

```
func (t *Controller) GetOnHoldBalanceWithOperationId(token_id string,
operation_id string) (interface{} error) {
    return t.Ctx.Hold.GetOnHoldBalanceWithOperationId(token_id,
operation_id)
}
```

Parameters:

- `token_id string` – The ID of the token.
- `operation_id string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, a JSON string indicating the holding balance.

Return Value Example:

```
{
    "holding_balance": 10,
    "msg": "Current Holding Balance of OperationId opr_121 for token
digiCurr101 is : 10"
}
```

### Methods for Token Behavior Management - Burnable Behavior

**BurnTokens**

This method deactivates, or burns, tokens from the transaction caller's account. The caller of this method must have an account and the burner role. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called by the `Account Owner` of the account with the burner role.

```
func (t *Controller) BurnTokens(token_id string, quantity float64)
(interface{} error) {
    tokenAssetValue, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, err
    }
    return t.Ctx.Token.Burn(quantity, tokenAssetValue.Interface())
}
```

Parameters:

- `token_id string` – The ID of the token.
- `quantity float64` – The number of tokens to burn.

Returns:

- On success, a success message with the quantity of tokens burned and the account ID.

Return Value Example:

```
{"msg":"Successfully burned 1 tokens from account id:
oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210c
706e (org_id : Org1MSP, user_id : user2)"}
```

### Custom Methods

You can use the token SDK methods to write custom methods for your business application.

Make sure to track the return value when you use the token SDK methods. Also, to avoid double-spending, do not combine multiple async functions that operate on the same key-value pairs in the state database. Instead, use the `BulkTransferTokens` method, to make multiple transfers in one method.

The following example shows how to use token SDK methods in custom methods. When the `BuyTicket` method is called, it transfers 20 tokens from the caller's account to the seller's account, and returns the transaction message of the transfer.

```
func (t *Controller) BuyTicket(TokenId string, SellerOrgId string,
SellerUserId string) (interface{}, error){
    token, err := t.Ctx.Token.Get(TokenId)
    if err != nil {
        return nil, err
    }

    /**
    * The following method t.Ctx.Account.GenerateAccountId(TokenId,
SellerOrgId, SellerUserId) generates account id of the seller
    */
```

```
    sellerAccountId, err := t.Ctx.Account.GenerateAccountId(TokenId,
SellerOrgId, SellerUserId)
    if err != nil {
        return nil, err
    }

    /**
    * The following method t.Ctx.Token.Transfer(sellerAccountId, 20,
token) transfers the quantity 20 from caller's
    * account & to seller's account.
    */
    transaction, err := t.Ctx.Token.Transfer(sellerAccountId, 20,
token)
    if err != nil {
        return nil, err
    }
    return transaction, nil
}
```

### Token SDK Methods

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Holdable Behavior
  - Burnable Behavior

### Methods for Access Control Management

The token SDK provides an access control function. Some methods can be called only by a `Token Admin`, `Org Admin` or `AccountOwner` of the token. You can use this feature to ensure that operations are carried out only by the intended users. Any unauthorized access results in an error. To use the access control function, import the `Authorization` class from the `../lib/auth` module.

```
import { Authorization } from '../lib/auth';
```

**`AddAdmin`**

This method adds a user as a `Token Admin` of the token chaincode.

```
Ctx.AddAdmin(org_id string, user_id string) (interface{}, error)
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- On success, a success message and details for the `Token Admin` user that was added. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"msg":"Successfully added Admin (Org_Id: Org1MSP, User_Id: user2)"}
```

**`RemoveAdmin`**
This method removes a user as a `Token Admin` of the token chaincode.

```
Ctx.RemoveAdmin(org_id string, user_id string) (interface{}, error)
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- On success, a success message and details for the `Token Admin` user that was removed. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"msg":"Successfuly removed Admin (Org_Id Org1MSP User_Id user1)"}
```

**`GetAllAdmins`**
This method returns a list of all users who are a `Token Admin` of the token chaincode.

```
Ctx.GetAllAdmins() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a list of all users who are a `Token Admin` of the token chaincode. On error, a non-nil error object containing an error message.

Return Value Example:

```
[{"OrgId":"Org1MSP","UserId":"admin"},{"OrgId":"Org1MSP","UserId":"user1"}]
```

**GetAllAdminUsers**

This method returns a list of all users who are a `Token Admin` of the token chaincode.

```
Ctx.Admin.GetAllAdminUsers() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a list of all users who are a `Token Admin` of the token chaincode in `map[string]interface{}` form. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"admins":[{"OrgId":"Org1MSP","UserId":"admin"},
{"OrgId":"Org1MSP","UserId":"user1"}]}
```

**CheckAuthorization**

Use this method to add access control to your chaincode. Many of the automatically generated token methods use access control. The mapping between the SDK receiver and the methods which have access control is described in the `oChainUtil.go` file. To use your own access control or to disable access control, remove the access control code from the automatically generated controller methods and custom methods.

```
var t TokenAccess
      var r RoleAccess
      var a AccountAccess
      var as AccountStatusAccess
      var h HoldAccess
      var ad AdminAccess
      var trx TransactionAccess
      var tc TokenConversionAccess
      var auth AuthAccess

      auth.IsTokenAdmin = []string{"Admin", "MultipleAccountOwner"}

      trx.DeleteHistoricalTransactions = []string{"Admin"}
      ad.AddAdmin = []string{"Admin"}
      ad.RemoveAdmin = []string{"Admin"}
      ad.GetAllAdmins = []string{"Admin", "OrgAdmin"}
      ad.AddOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
      ad.RemoveOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
      ad.GetOrgAdmins = []string{"Admin", "OrgAdmin"}
      ad.IsTokenAdmin = []string{"Admin", "MultipleAccountOwner",
"OrgAdmin"}
      t.Save = []string{"Admin"}
      t.GetAllTokens = []string{"Admin", "OrgAdmin"}
      t.Update = []string{"Admin"}
```

```
      t.GetTokenDecimals = []string{"Admin", "OrgAdmin"}
      t.GetTokensByName = []string{"Admin", "OrgAdmin"}
      t.AddRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
      t.RemoveRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
      t.IsInRole = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      t.GetTotalMintedTokens = []string{"Admin", "OrgAdmin"}
      t.GetNetTokens = []string{"Admin", "OrgAdmin"}
      t.Get = []string{"Admin", "OrgAdmin"}
      t.GetTokenHistory = []string{"Admin", "OrgAdmin"}
      a.CreateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      a.AssociateToken = []string{"Admin", "OrgAdminAccountIdCheck"}
      a.GetAllAccounts = []string{"Admin"}
      a.GetAllOrgAccounts = []string{"Admin", "OrgAdminOrgIdCheck"}
      a.GetAccount = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      a.History = []string{"Admin", "AccountOwner"}
      a.GetAccountTransactionHistory = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetAccountTransactionHistoryWithFilters = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetSubTransactionsById = []string{"Admin", "TransactionInvoker"}
      a.GetSubTransactionsByIdWithFilters = []string{"Admin",
"TransactionInvoker"}
      a.GetAccountBalance = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      a.GetAccountOnHoldBalance = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetOnHoldIds = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      a.GetAccountsByUser = []string{"Admin", "OrgAdminOrgIdCheck",
"MultipleAccountOwner"}

      as.Get = []string{"Admin", "OrgAdminAccountIdCheck", "AccountOwner"}
      as.ActivateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      as.SuspendAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      as.DeleteAccount = []string{"Admin", "OrgAdminOrgIdCheck"}

      r.GetAccountsByRole = []string{"Admin"}
      r.GetUsersByRole = []string{"Admin"}
      r.GetOrgAccountsByRole = []string{"Admin", "OrgAdminOrgIdCheck"}
      r.GetOrgUsersByRole = []string{"Admin", "OrgAdminOrgIdCheck"}

      tc.InitializeExchangePoolUser = []string{"Admin"}
      tc.AddConversionRate = []string{"Admin"}
      tc.UpdateConversionRate = []string{"Admin"}
      tc.GetConversionRate = []string{"Admin", "OrgAdmin", "AnyAccountOwner"}
      tc.GetConversionRateHistory = []string{"Admin", "OrgAdmin",
"AnyAccountOwner"}
```

```
tc.TokenConversion = []string{"Admin", "AnyAccountOwner"}
tc.GetExchangePoolUser = []string{"Admin"}
```

```
Ctx.Auth.CheckAuthorization(classFuncName string, args ...string)
(bool, error)
```

Parameters:

- `classFuncName string` – The map value between the receivers and methods as described in the `oChainUtil.go` file.
- `args` – A variable argument, where `args[0]` is the constant `TOKEN` and `args[1]` is the `account_id` argument, if required.

Returns:

- A Boolean response and an error message if an error is encountered.

**`IsUserTokenAdmin`**
This method returns the Boolean value `true` if the caller of the function is a `Token Admin`. Otherwise the method returns `false`.

```
Ctx.Auth.IsUserTokenAdmin()  (bool, error)
```

Parameters:

- `user_id` – The user name or email ID of the user.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.

Returns:

- A Boolean response and an error message if an error is encountered.

Return Value Example:

```
{"result":false}
```

**`AddOrgAdmin`**
This method adds a user as an `Org Admin` of the organization.

```
Ctx.Admin.AddOrgAdmin(org_id, user_id) (interface{}, error)
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully added Org Admin (Org_Id: Org1MSP, User_Id:
orgAdmin)"
}
```

**RemoveOrgAdmin**

This method removes a user as an `Org Admin` of an organization.

```
Ctx.Admin.RemoveOrgAdmin(org_id, user_id) (interface{}, error)
```

Parameters:

- `org_id string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as an `Org Admin` of the organization.

Return Value Example:

```
{
    "msg": "Successfully removed Org Admin (Org_Id Org1MSP User_Id orgAdmin)"
}
```

**GetOrgAdmins**

This method returns a list of all users who are an `Org Admin` of an organization.

```
Ctx.Admin.GetAllOrgAdmins() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a JSON list that includes `OrgId` and `UserId` objects.

Return Value Example:

```
{
    "admins": [
        {
            "OrgId": "Org1MSP",
```

```
            "UserId": "orgadmin"
        },
        {
            "OrgId": "Org1MSP",
            "UserId": "orgadmin1"
        },
        {
            "OrgId": "Org1MSP",
            "UserId": "orgadmin2"
        }
    ]
}
```

### Methods for Token Configuration Management

**`GetTokenDecimals`**

This method returns the number of decimal places available for a fractional token. If the `divisible` behavior is not specified, then the default value is 0.

```
Ctx.Token.GetTokenDecimals(token_id string) (int, error)
```

Parameters:

- none

Returns:

- On success, the decimal places of the token, in the number data type. On error, a non-nil error object containing an error message.

Return Value Example:

```
1
```

**`GetAllTokens`**

This method returns all of the token assets that are saved in the state database. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.Token.GetAllTokens()  (interface{}, error)
```

Parameters:

- none

Returns:

- On success, an array of a map of all token assets. On error, a non-nil error object containing an error message.

Return Value Example:

```
"payload": [
    {
        "key": "t1",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "Currency_name value",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "Token_desc value",
            "Token_id": "t1",
            "Token_name": "obptok",
            "Token_to_currency_ratio": 999,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```

**GetTokensByName**

This method returns all of the token assets with the specified name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.Token.GetTokensByName(token_name string) (interface{}, error)
```

Parameters:

- `token_name string` – The name of the token, which corresponds to the `Token_name` property of the model. The value is the class name of the token.

Returns:

- It returns an array of a map of all of the token assets of the specified name.

Return Value Example:


```
"payload": [
    {
        "key": "t1",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "Currency_name value",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "Token_desc value",
            "Token_id": "t1",
            "Token_name": "obptok",
            "Token_to_currency_ratio": 999,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    },
    {
        "key": "obp2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "holdable",
                "burnable",
                "roles"
            ],
            "Currency_name": "",
            "Divisible": {
                "Decimal": 8
            },
            "Mintable": {
                "Max_mint_quantity": 10000
```

```
            },
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter",
                "notary_role_name": "notary"
            },
            "Token_desc": "",
            "Token_id": "obp2",
            "Token_name": "obptok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```

**Get**

This method returns a token object if it is present in the state database. This method can be called only by a `Token Admin` of the token chaincode.

```
Ctx.Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token to return.
- `result` – A variable argument, where the first argument `result[0]` is a reference of an empty `Token` object of the required type.

Returns:

- On success, a map with the token asset data. The variable argument `result[0]` contains the token data. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "Digital Currency equiv of dollar",
    "Token_type": "fungible",
    "Behavior": ["divisible", "mintable", "transferable", "burnable",
"holdable", "roles"],
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter",
        "notary_role_name": "notary"
    },
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "Divisible": {
```

```
        "Decimal": 1
    },
    "Token_to_currency_ratio": 1,
    "Currency_representation": "DOLLAR"
}
```

**IsTokenType**

This method tests whether a token asset exists for a specified token ID.

```
Ctx.Model.IsTokenType(token_id: string) error
```

Parameters:

- `token_id: string` – The ID of the token to check.

Returns:

- If a token asset exists with the specified ID, a nil error. Otherwise, a non-nil error object containing an error message.

Return Value Example:

```
nil
```

**Save**

This method creates a token and saves its properties in the state database.

```
Ctx.Token.Save(args ...interface{}) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token to return.
- `args` – A variable argument, where the first argument `args[0]` is a reference of the token `struct` data of the required type to add to the ledger.

Returns:

- On success, an `interface{}` object with details about the token that was saved to the state database. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "",
    "Token_type": "fungible",
    "Behavior": ["divisible", "mintable", "transferable", "burnable",
"roles"],
    "Roles": {
        "minter_role_name": "minter"
```

```
    },
    "Mintable": {
        "Max_mint_quantity": 1000
    },
    "Divisible": {
        "Decimal": 2
    },
    "Currency_name": "",
    "Token_to_currency_ratio": 1
}
```

**Update**

This method updates token properties. After a token asset is created, you can update only the `token_desc` value and its custom properties.

```
Ctx.Token.Update(args ...interface{}) (interface{}, error)
```

Parameters:

- An asset that contains a reference to the token `struct` data of required type to update in the ledger.

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Token_id": "digiCurr101",
    "Token_name": "digicur",
    "Token_desc": "Digital Currency equiv of dollar",
    "Token_type": "fungible",
    "Behavior": ["divisible", "mintable", "transferable", "burnable",
"roles"],
    "Roles": {
        "minter_role_name": "minter"
    },
    "Mintable": {
        "Max_mint_quantity": 1000
    },
    "Divisible": {
        "Decimal": 2
    },
    "Currency_name": "",
    "Token_to_currency_ratio": 1
}
```

**GetByRange**

This method calls the fabric `getStateByRange` method internally. Even though any asset with the given ID is returned from the ledger, this method casts the asset into the caller Asset type.

```
Ctx.Token.GetByRange(startId string, endId string,
asset ...interface{}) ([]map[string]interface{}, error)
```

Parameters:

- `startId: string` – The starting key of the range. This key is included in the range.
- `endId: string` – The end key of the range. This key is excluded from the range.
- `asset[0]` – An empty slice of the token of the required type. If the method runs successfully, this contains the requested result.

Returns:

- On success, a slice of maps containing the token asset details for tokens where the `token_id` value is in the specified range. On error, a non-nil error object containing an error message.

Return Value Example:

```
[{
    "Key":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0
526bc056c4b8974",
    "Record": {
        "AccountId":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0
526bc056c4b8974",
        "AssetType": "oaccount",
        "Balance": 99,
        "BalanceOnHold": 1,
        "BapAccountVersion": 0,
        "OrgId": "Org1MSP",
        "TokenId": "t1",
        "TokenName": "loyaltok123",
        "UserId": "u1"
    }
}, {
    "Key":
"oaccount~loyaltok123~ac30c5ca924a2c7def61acf596d91e0cca70bc8cd233179df
4efb2791b56336b",
    "Record": {
        "AccountId":
"oaccount~loyaltok123~ac30c5ca924a2c7def61acf596d91e0cca70bc8cd233179df
4efb2791b56336b",
        "AssetType": "oaccount",
        "Balance": 0,
        "BalanceOnHold": 0,
```

```
        "BapAccountVersion": 0,
        "OrgId": "Org1MSP",
        "TokenId": "t1",
        "TokenName": "loyaltok123",
        "UserId": "u2"
    }
}, {
    "Key":
"oaccount~loyaltok123~aef96c40d99e09ef17f9bdda7038e8fbe829a327bae2b4d8d9fcf75
2190f3ff0",
    "Record": {
        "AccountId":
"oaccount~loyaltok123~aef96c40d99e09ef17f9bdda7038e8fbe829a327bae2b4d8d9fcf75
2190f3ff0",
        "AssetType": "oaccount",
        "Balance": 0,
        "BapAccountVersion": 0,
        "BalanceOnHold": 0,
        "OrgId": "Org1MSP",
        "TokenId": "t1",
        "TokenName": "loyaltok123",
        "UserId": "admin"
    }
}, {
    "Key": "oadmin~Org1MSP~admin",
    "Record": {
        "AssetType": "oadmin",
        "Key": "oadmin~Org1MSP~admin",
        "OrgId": "Org1MSP",
        "UserId": "admin"
    }
}, {
    "Key": "ohold~loyaltok123~t1~op1",
    "Record": {
        "AssetType": "ohold",
        "FromAccountId":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0526bc0
56c4b8974",
        "HoldingId": "ohold~loyaltok123~t1~op1",
        "NotaryAccountId":
"oaccount~loyaltok123~ac30c5ca924a2c7def61acf596d91e0cca70bc8cd233179df4efb27
91b56336b",
        "OperationId": "op1",
        "Quantity": 1,
        "TimeToExpiration": "0",
        "ToAccountId":
"oaccount~loyaltok123~aef96c40d99e09ef17f9bdda7038e8fbe829a327bae2b4d8d9fcf75
2190f3ff0",
        "TokenId": "t1",
        "TokenName": "loyaltok123"
    }
}, {
    "Key": "ometadata~loyaltok123~t1",
```

```
    "Record": {
        "AssetType": "ometadata",
        "Metadata_id": "ometadata~loyaltok123~t1",
        "Token_id": "t1",
        "Token_name": "loyaltok123",
        "Total_minted_amount": 100,
        "Total_supply": 100
    }
}, {
    "Key":
"orole~t1~minter~oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527
233bcf9121ff95b0526bc056c4b8974",
    "Record": {
        "AccountID":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0
526bc056c4b8974",
        "AssetType": "orole",
        "Key":
"orole~t1~minter~oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527
233bcf9121ff95b0526bc056c4b8974",
        "RoleName": "minter",
        "TokenId": "t1"
    }
}, {
    "Key":
"otransaction~4a774f6493f6521cab9eda96822cb3bb4103c0738ee2dbb9a193b868a
ce36fa5",
    "Record": {
        "Amount": 100,
        "AssetType": "otransaction",
        "FromAccountId": "",
        "HoldingId": "",
        "NumberOfSubTransactions": 0,
        "Timestamp": "2021-08-25T23:04:42+05:30",
        "ToAccountId":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0
526bc056c4b8974",
        "TokenId": "t1",
        "TransactionId":
"otransaction~4a774f6493f6521cab9eda96822cb3bb4103c0738ee2dbb9a193b868a
ce36fa5",
        "TransactionType": "MINT"
    }
}, {
    "Key":
"otransaction~69f3cefbcb64b73f01a0eadff87169f456873ccebe61ca8da3eef3f46
5f0c129",
    "Record": {
        "Amount": 1,
        "AssetType": "otransaction",
        "FromAccountId":
"oaccount~loyaltok123~a4bd3d8abfb1708198971311df77bb527233bcf9121ff95b0
526bc056c4b8974",
```

```
        "HoldingId": "ohold~loyaltok123~t1~op1",
        "NumberOfSubTransactions": 0,
        "Timestamp": "2021-08-25T23:06:13+05:30",
        "ToAccountId":
"oaccount~loyaltok123~aef96c40d99e09ef17f9bdda7038e8fbe829a327bae2b4d8d9fcf75
2190f3ff0",
        "TokenId": "t1",
        "TransactionId":
"otransaction~69f3cefbcb64b73f01a0eadff87169f456873ccebe61ca8da3eef3f465f0c12
9",
        "TransactionType": "ONHOLD"
    }
}, {
    "Key": "t1",
    "Record": {
        "AssetType": "otoken",
        "Behavior": ["divisible", "mintable", "transferable", "burnable",
"holdable", "roles"],
        "Currency_Name": "a",
        "Divisible": {
            "Decimal": 2
        },
        "Effective_From_Date": "2020-09-09T00:00:00Z",
        "Mintable": {
            "Max_mint_quantity": 10000
        },
        "Roles": {
            "minter_role_name": "minter"
        },
        "Token_To_Currency_Ratio": 1,
        "Token_desc": "",
        "Token_id": "t1",
        "Token_name": "loyaltok123",
        "Token_type": "fungible"
    }
}]
```


**History**

This method returns the token history for a specified token ID.

```
Ctx.Token.History(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON array that represents the token history.

Return Value Example:


```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-09-01T16:46:33Z",
        "TxId":
"12333b8a4f63aa9b3a34072efcbd7df546c6d1e7d82a7a9596e899383656d6f7",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "Currency_name1": "",
            "Divisible": {
                "Decimal": 2
            },
            "Mintable": {
                "Max_mint_quantity": 1000
            },
            "Roles": {
                "minter_role_name": "minter"
            },
            "Token_desc": "updated description",
            "Token_id": "token",
            "Token_name": "fiatmoneytok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-09-01T16:04:25Z",
        "TxId":
"99702e2dad7554a5ee4716a0d01d3e394cbce39bea8bade265d8911f30ebad0b",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "Currency_name1": "",
            "Divisible": {
                "Decimal": 2
            },
```

```
            "Mintable": {
                "Max_mint_quantity": 1000
            },
            "Roles": {
                "minter_role_name": "minter"
            },
            "Token_desc": "",
            "Token_id": "token",
            "Token_name": "fiatmoneytok",
            "Token_to_currency_ratio": 0,
            "Token_type": "fungible",
            "Token_unit": "fractional"
        }
    }
]
```

### Methods for Account Management

**`GenerateAccountId`**

This method returns an account ID, which is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`org_id`) of the user in the current network organization and the unique token ID (`token_id`).

```
Ctx.Account.GenerateAccountId(token_id string, org_id string, user_id
string) (string, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, the generated account ID. On error, a rejection with an error message.

Return Value Example:

```
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f
```

**`CreateAccount`**

This method creates an account for a specified user and token. Every user who has tokens at any point must have an account. Accounts track a user's balance, on-hold balance, and transaction history. An account ID is an alphanumeric set of characters, prefixed with `oaccount~<token asset name>~` and followed by a hash of the user name or email ID (`user_id`) of the instance owner or the user who is logged in to the instance, the membership

service provider ID (`org_id`) of the user in the current network organization. This method can be called only by the `Token Admin` of the chaincode.

```
t.Ctx.Account.CreateAccount(org_id string, user_id string, token_type
string)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.
- `token_type: string` – The type of the token, which must be `fungible`.

Returns:

- On success, the account object that was created. On error, a rejection with an error message.

Return Value Example:

```
{
 "AssetType":"oaccount",
"AccountId":"oaccount~a73085a385bc96c4a45aa2dff032e7dede82c0664dee5f396
b7c5854eeafd4bd",
   "UserId":"user1",
   "OrgId":"Org1MSP",
   "BapAccountVersion": 0,
   "AccountType":"fungible",
   "TokenId":"",
   "TokenName":"",
   "Balance":0,
   "BalanceOnHold":0
}
```

**`AssociateToken`**

This method associates a fungible token with an account. This method can be called only by a `Token Admin` of the chaincode.

```
t.Ctx.Account.AssociateToken(account_id, token_id)
```

Parameters:

- `account_id string` – The ID of the account.
- `token_id string` – The ID of the token.

Returns:

- On success, a JSON object of the updated account.

Return Value Example:

```
{
"AssetType":"oaccount",
"AccountId":"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54b
dbebf48eb",
"BapAccountVersion": 0,
"UserId":"admin",
"OrgId":"Org1MSP",
"AccountType":"fungible",
"TokenId":"token1",
"TokenName":"loyaltok",
"Balance":0,
"BalanceOnHold":0
}
```

**GetAccountWithStatus**

This method returns account details for a specified account, including account status.

```
Ctx.Account.GetAccountWithStatus(account_id string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, the requested account details. On error, a rejection with an error message.

Return Value Example:

```
{
  "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
  "AssetType": "oaccount",
  "Balance": 95,
  "BalanceOnHold": 0,
  "BapAccountVersion": 8,
  "OrgId": "appdev",
  "Status": "active",
  "TokenId": "obp1",
  "TokenName": "obptok",
  "TokenType": "fungible",
  "UserId": "idcqa"
}
```

**GetAccount**

This method returns account details for a specified account.

```
Ctx.Account.GetAccount(account_id string) (Account, error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, the requested account details. On error, a rejection with an error message.

Return Value Example:

```
{
    "AssetType": "oaccount",
    "BapAccountVersion": 0,
    "AccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "UserId": "user1",
    "OrgId": "Org1MSP",
    "TokenId": "digiCurr101",
    "TokenName": "digicur",
    "Balance": 0,
    "BalanceOnHold": 0
}
```

**GetAccountHistory**

This method returns an array of the account history details for a specified account.

```
Ctx.Account.History(account_id string) ([]interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a `map[string]interface{}` array that contains the account history details. The account data is shown under the `Value` key in the map. On error, a non-nil error object containing an error message. The return value is the same as the GetAccountHistory method.

Return Value Example:

```
[
  {
      "IsDelete": "false",
      "Timestamp": "2023-08-28T19:31:15Z",
      "TxId":
"adde470a63860ec1013bd5c5987e8a506a48942a91b0f39fc8e561374042bd27",
      "Value": {
          "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
          "AssetType": "oaccount",
          "Balance": 100,
```

```
            "BalanceOnHold": 0,
            "BapAccountVersion": 1,
            "OrgId": "Org1MSP",
            "TokenId": "t1",
            "TokenName": "obptok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-08-28T19:30:23Z",
        "TxId":
"8fbeda2ba60ba175091faae5ae369247775f2cba45c4d6d1ead6f0b05be84743",
        "Value": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "Org1MSP",
            "TokenId": "t1",
            "TokenName": "obptok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-08-28T19:29:54Z",
        "TxId":
"19bb296ae71709e91b097ba5d9ebd7f7522095880382fbf5913334a46a6026aa",
        "Value": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "Org1MSP",
            "TokenId": "",
            "TokenName": "",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```

**GetAccountOnHoldBalance**

This method returns the on-hold balance for a specified account.

```
Ctx.Account.getAccountOnHoldBalance(account_id: string)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise object with the current on-hold balance and a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
   "holding_balance":0,
   "msg":"Total Holding Balance of Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id: Org1MSP, user_id: user1) is 0"
}
```

**GetAllAccounts**

This method returns a list of all accounts. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.func (t *Controller) GetAllAccounts() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a JSON array that lists all accounts.

Return Value Example:

```
"payload": [
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
        "valueJson": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
            "AssetType": "oaccount",
            "Balance": 100,
            "BalanceOnHold": 0,
            "BapAccountVersion": 1,
```

```
            "OrgId": "appdev",
            "TokenId": "t1",
            "TokenName": "obptok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```

**GetUserByAccountId**
This method returns the user details for a specified account.

```
Ctx.Account.GetUserByAccountById(account_id string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, a promise with a JSON object that includes three properties:
  - `user_id` – The user name or email ID of the user.
  - `org_id` – The membership service provider (MSP) ID of the user in the current network organization.
  - `token_id` – The ID of the token.
- On error, a non-nil error object containing an error message.

Return Value Example:

```
{
   "org_id":"Org1MSP",
   "token_id":"digiCurr101",
   "user_id":"user1"
}
```

**GetAccountBalance**
This method returns the account balance for a specified account.

```
Ctx.GetAccountBalance(token_id string, org_id string, user_id string)
(interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- On success, an interface with a message string and the current balance. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "msg": "Current Balance of +p2uaMTsU9D74l9XpHQ2c55ic/
2gbO4NZITC4Zq4P8E= is: 200",
    "user_balance": 200
}
```

**GetAllOrgAccounts**

This method returns a list of all token accounts that belong to a specified organization.

```
Ctx.Account.GetAllOrgAccounts(org_id string) (interface{}, error)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts for the specified organization.

Return Value Example:

```
[
    {
        "key":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
        "valueJson": {
            "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
            "AssetType": "oaccount",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "appdev",
            "TokenId": "token",
            "TokenName": "fiatmoneytok",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "key":
"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a
850",
        "valueJson": {
            "AccountId":
"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a
850",
            "AssetType": "oaccount",
```

```
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "OrgId": "appdev",
            "TokenId": "token",
            "TokenName": "fiatmoneytok",
            "TokenType": "fungible",
            "UserId": "example_minter"
        }
    }
]
```

### Methods for Role Management

**`AddRoleMember`**

This method adds a role to a specified user and token.

```
Ctx.Token.AddRoleMember(role string, account_id string, tokenAsset
interface{}) (interface{}, error)
```

Parameters:

- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `account_id: number` – The account ID to add the role to.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, it returns a map with a success message indicating the addition of the role to the account.. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
   "msg":"Successfully added role minter to
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1)"
}
```

**`RemoveRoleMember`**

This method removes a role from a specified user and token.

```
Ctx.Token.RemoveRoleMember(role string, account_id string, tokenAsset
interface{}) (interface{}, error)
```

Parameters:

- `role: string` – The name of the role to remove from to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file. Similarly, the `notary` role corresponds to the `notary_role_name` property of the specification file.
- `account_id: number` – The account ID to remove the role from.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, it returns a map with a success message indicating the removal of the role from the account.. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
  "msg":"successfully removed member_id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1) from role minter"
}
```

**GetAccountsByRole**

This method returns a list of all accounts for a specified role and token.

```
Ctx.Role.GetAccountsByRole(token_id string, user_role string)
(interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"accounts":
["oaccount~obptok~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f"]}
```

**GetUsersByRole**

This method returns a list of all users for a specified role and token.

```
Ctx.Role.GetUsersByRole(token_id string, user_role string)
(interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of user objects. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "Users": [
        {
            "token_id":"digiCurr101",
            "user_id": "user1",
            "org_id": "Org1MSP"
        }
    ]
}
```

**`IsInRole`**
This method indicates whether a user and token has a specified role.

```
Ctx.Token.IsInRole(role string, account_id string, tokenAsset interface{})
(bool, error)
```

Parameters:

- `role: string` – The name of the role to check.
- `account_id: number` – The account ID to check.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, it returns a map with a success message indicating the removal of the role from the account.. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "result": false
}
```

**`RoleCheck`**
This method checks if the provided account ID is a member of any role.

```
Ctx.Token.RoleCheck(account_id string, tokenAsset interface{}) (bool, error)
```

Parameters:

- `account_id: string` – The account ID to check.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- If the specified account has any role, a success message and the Boolean value `true`. Otherwise, the Boolean value `false`. On error, a non-nil error object containing an error message.

Return Value Example:

```
{ result: true }
```

**GetOrgUsersByRole**
This method returns information about all users that have a specified role in a specified organization.

```
Ctx.Role.GetOrgUsersByRole(token_id string, user_role string, org_id
string) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all users with the specified role in the specified organization.

Return Value Example:

```
{
    "Users": [
        {
            "org_id": "Org1MSP",
            "token_id": "token",
            "user_id": "admin"
        },
        {
            "org_id": "Org1MSP",
            "token_id": "token",
            "user_id": "orgAdmin"
        }
    ]
}
```

**GetOrgAccountsByRole**
This method returns information about all accounts that have a specified role in a specified organization.

```
Ctx.Role.GetOrgAccountsByRole(token_id string, user_role string, org_id
string) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `role: string` – The name of the role to check for.
- `org_id: string` – The membership service provider (MSP) ID of the organization.

Returns:

- On success, a list of all accounts with the specified role in the specified organization.

Return Value Example:

```
{
    "accounts": [

"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",

"oaccount~9c650574af9025a6106c8d12a801b079eda9ae2e3399fc2fbd5bd683d738a850"
    ]
}
```

### Methods for Transaction History Management

**GetAccountTransactionHistory**
This method returns an array of the transaction history details for a specified account.

```
Ctx.Account.GetAccountTransactionHistory(account_id string) (interface{},
error)
```

Parameters:

- `account_id: string` – The ID of the account.

Returns:

- The return value is the same as the GetAccountTransactionHistory method.
- On success, an array of JSON account transaction objects.
- On error, a non-nil error object containing an error message.

Return Value Example:

```
[
  {
      "NumberOfSubTransactions": 2,
```

```
      "balance": 160,
      "onhold_balance": 0,
      "timestamp": "2023-09-06T06:51:48Z",
      "token_id": "t1",
      "transacted_amount": 20,
      "transaction_id":
"otransaction~bd3e8d7d0bcdbed0469a2fccfe95f7ebbeb1987d8385bccf5c84bf802
51e748c",
      "transaction_type": "BULKTRANSFER"
  },
  {
      "balance": 180,
      "onhold_balance": 0,
      "timestamp": "2023-09-06T06:47:14Z",
      "token_id": "t1",
      "transacted_account":
"oaccount~692a7465c01e36b694cb8ae86e6c6584240aa1f865fde54f95f32429eadd4
097",
      "transacted_amount": 10,
      "transaction_id":
"otransaction~250996f1df6a36a1b647f522efcaaf48fd70452d711c247fc4cd475b8
e752b08",
      "transaction_type": "DEBIT"
  },
  {
      "balance": 190,
      "onhold_balance": 0,
      "timestamp": "2023-09-06T06:47:08Z",
      "token_id": "t1",
      "transacted_account":
"oaccount~bb5a0b57d895327c8a8cd1f267310cbf3ae542bc854fab8188b5083a969d7
2fb",
      "transacted_amount": 10,
      "transaction_id":
"otransaction~664325a25ae6b19b23693c66f83811184e0a78fabb49122359a2dbf20
9f32976",
      "transaction_type": "DEBIT"
  },
  {
      "balance": 200,
      "onhold_balance": 0,
      "timestamp": "2023-09-06T06:46:46Z",
      "token_id": "t1",
      "transacted_account":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
      "transacted_amount": 100,
      "transaction_id":
"otransaction~7f49564b1eb61d4c8be0ef61cd5e635b533ca533907944e4ec500f390
237fd6b",
      "transaction_type": "MINT"
  },
  {
```

```
      "balance": 100,
      "onhold_balance": 0,
      "timestamp": "2023-08-28T19:31:15Z",
      "token_id": "t1",
      "transacted_account":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
      "transacted_amount": 100,
      "transaction_id":
"otransaction~adde470a63860ec1013bd5c5987e8a506a48942a91b0f39fc8e561374042bd2
7",
      "transaction_type": "MINT"
  }
]
```

**GetAccountTransactionHistoryWithFilters**

This method returns an array of the transaction history details for a specified transaction. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
t.Ctx.Account.GetAccountTransactionHistoryWithFilters (transaction_id:
string, filters?: SubTransactionFilters)
```

Parameters:

- `Transaction_id: string` – The ID of the transaction.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetAccountTransactionHistoryWithFilters 'token1' 'appbuilder12'
'user_minter'
'{"PageSize":10,"Bookmark":"1","StartTime":"2022-01-25T17:41:42Z","EndTime":"20
22-01-25T17:59:10Z"}'
```

```
[
    {
        "balance": 90,
        "onhold_balance": 0,
        "timestamp": "2022-04-20T19:43:36Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~7a4d67118e623a876b77c67e76b819269a8d4a509aece5d2263fb274a9beb3b8",
        "transacted_amount": 5,
        "transaction_id":
"otransaction~dd9986d3686e52264935558e42026fbf8a9af48b06a3256a58b453f5ada4e63
6",
        "transaction_type": "DEBIT"
    },
```

```
    {
        "balance": 95,
        "onhold_balance": 0,
        "timestamp": "2022-04-20T19:43:22Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~0642308fc4c514c257ebf04326c63f990e2531bfd59d0b952056094da61e0
4ab",
        "transacted_amount": 5,
        "transaction_id":
"otransaction~5e53424de3d691cf6b2a55ea3dc478c555d8784111c11847e594194d6
c2e7755",
        "transaction_type": "DEBIT"
    },
    {
        "balance": 100,
        "onhold_balance": 0,
        "timestamp": "2022-04-20T19:42:54Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~b63935592a702d30bedb87ae97b9b1ba7d0f346716adc4f5a4192220bf410
d4e",
        "transacted_amount": 100,
        "transaction_id":
"otransaction~94c467825ce9f66cc69958d38b169022a69eebc66b75b7d6e0b0585af
2c3c228",
        "transaction_type": "MINT"
    }
]
```


**GetSubTransactionsById**

This method returns an array of the transaction history details for a specified transaction.

```
t.Ctx.Account.GetSubTransactionsById(transaction_id string)
```

Parameters:

- `transaction_id: string` – The ID of the transaction.

Example:

```
ochain invoke GetSubTransactionsById
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f8
64b9b'
```


```
[
    {
        "balance": 80,
        "onhold_balance": 0,
        "timestamp": "2022-04-21T05:02:33Z",
        "token_id": "tokenId",
        "transacted_account":
```

```
"oaccount~7a4d67118e623a876b77c67e76b819269a8d4a509aece5d2263fb274a9beb3b8",
        "transacted_amount": 5,
        "transaction_id":
"otransaction~33de5d63058d5e9abc011bc850878dfb7ac3080495729aed345c45b2f21735f
a~c81e728d9d4c2f636f067f89cc14862c",
        "transaction_type": "DEBIT"
    },
    {
        "balance": 85,
        "onhold_balance": 0,
        "timestamp": "2022-04-21T05:02:33Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~0642308fc4c514c257ebf04326c63f990e2531bfd59d0b952056094da61e04ab",
        "transacted_amount": 5,
        "transaction_id":
"otransaction~33de5d63058d5e9abc011bc850878dfb7ac3080495729aed345c45b2f21735f
a~c4ca4238a0b923820dcc509a6f75849b",
        "transaction_type": "DEBIT"
    }
]
```

**GetSubTransactionsByIdWithFilters**

This method returns an array of the transaction history details for a specified transaction.

```
t.Ctx.Account.GetSubTransactionsByIdWithFilters(transaction_id string,
filters ...SubTransactionFilters)
```

Parameters:

- `transaction_id: string` – The ID of the transaction.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Example:

```
ochain invoke GetSubTransactionsByIdWithFilters
'otransaction~21972b4d206bd52ea77924efb259c67217edb23b4386580d1bee696f6f864b9b'
'{"PageSize":10,"Bookmark":"1"}'
```

```
[
    {
        "balance": 80,
        "onhold_balance": 0,
        "timestamp": "2022-04-21T05:02:33Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~7a4d67118e623a876b77c67e76b819269a8d4a509aece5d2263fb274a9beb3b8",
        "transacted_amount": 5,
```

```
        "transaction_id":
"otransaction~33de5d63058d5e9abc011bc850878dfb7ac3080495729aed345c45b2f
21735fa~c81e728d9d4c2f636f067f89cc14862c",
        "transaction_type": "DEBIT"
    },
    {
        "balance": 85,
        "onhold_balance": 0,
        "timestamp": "2022-04-21T05:02:33Z",
        "token_id": "tokenId",
        "transacted_account":
"oaccount~0642308fc4c514c257ebf04326c63f990e2531bfd59d0b952056094da61e0
4ab",
        "transacted_amount": 5,
        "transaction_id":
"otransaction~33de5d63058d5e9abc011bc850878dfb7ac3080495729aed345c45b2f
21735fa~c4ca4238a0b923820dcc509a6f75849b",
        "transaction_type": "DEBIT"
    }
]
```

**GetTransactionById**

This method returns the history of a `Transaction` asset.

```
t.Ctx.Transaction.GetTransactionById(transaction_id string)
```

Parameters:

- `transaction_id string` – The ID of the transaction asset.

Return Value Example:

```
{
    "history": [
        {
            "IsDelete": "false",
            "Timestamp": "2021-08-16 20:19:05.028 +0530 IST",
            "TxId":
"67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb2220",
            "Value": {
                "Amount": 3,
                "AssetType": "otransaction",
                "FromAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
                "HoldingId": "ohold~digicur~digiCurr101~op2",
                "NumberOfSubTransactions": 0,
                "Timestamp": "2021-08-16T20:19:05+05:30",
                "ToAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
                "TokenId": "digiCurr101",
```

```
                "TransactionId":
"otransaction~67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb222
0",
                "TransactionType": "RELEASEHOLD"
            }
        }
    ],
    "transaction_id":
"otransaction~67042154a6853011d111b13f73943f06d2a6ae3cfb9a84cb104482c359eb222
0"
}
```

**DeleteHistoricalTransactions**

This method deletes older transactions from the state database.

```
func (t *Controller) DeleteHistoricalTransactions(timestamp string)
(interface{}, error)
```

Parameters:

- `time_to_expiration: Date` – A time stamp that indicates when to delete transactions. Transaction assets that are older than the specified time will be deleted..

Return Value Example:

```
"payload": {
    "msg": "Successfuly deleted transaction older than
date:2021-08-18T05:43:30Z",
    "transactions": [

"otransaction~57d81f681aa215bb73d6c017d16be8b283d3fcb50051c85891a97d1d407fc34
2"
    ]
}
```

### Methods for Token Behavior Management - Mintable Behavior

**Mint**

This method mints tokens, which are then owned by the caller of the method. The caller must have an account and the minter role. The number of tokens that can be minted is limited by the `max_mint_quantity` property of `mintable` behavior in the specification file. If the `max_mint_quantity` property is not specified, an unlimited number of tokens can be minted. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account with the minter role.

```
Ctx.Token.Mint(quantity float64, tokenAsset interface{}) (interface{}, error)
```

Parameters:

- `quantity: number` – The number of tokens to mint.

- `tokenAsset` – The reference to the token asset to mint.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
  "msg":"Successfully minted 1000 tokens to Account Id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (Org-Id: Org1MSP, User-Id: admin)"
}
```

**`GetTotalMintedTokens`**
This method returns the total number of tokens minted.

```
Ctx.Token.GetTotalMintedTokens(tokenAsset interface{})
(map[string]interface{}, error)
```

Parameters:

- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message and a map of total minted tokens in the number data type. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"msg":"total minted amount for token with id digiCurr101 is
0","quantity":0}
```

**`GetNetTokens`**
This method returns the net quantity of tokens that are available in the system for a specified token. The net tokens are the amount of tokens remaining after tokens are burned. In equation form: net tokens = total minted tokens - total burned tokens. If no tokens are burned, then the number of net tokens is equal to the total minted tokens.

```
Ctx.Token.GetNetTokens(tokenAsset interface{})
(map[string]interface{}, error)
```

Parameters:

- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message and a map of the net quantity of tokens in the number data type. On error, an error message.

Return Value Example:

```
{"msg":"net minted amount for token with id digiCurr101 is 0","quantity":0}
```

**GetMaxMintQuantity**
This method returns the maximum mintable quantity for a token. If the `max_mint_quantity` behavior is not specified, then the default value is 0, which allows any number of tokens to be minted.

```
Ctx.Token.GetMaxMintQuantity(token_id string) (float64, error)
```

Parameters:

- `token_id: string` – The token ID to check.

Returns:

- On success, the maximum mintable quantity of the token, in the number data type. On error, a non-nil error object containing an error message.

Return Value Example:

```
20000
```

### Methods for Token Behavior Management - Transferable Behavior

**Transfer**
This method transfers tokens from the caller to a specified account. The caller of the method must have an account. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. This method can be called only by the `AccountOwner` of the account.

```
Ctx.Token.Transfer(to_account_id string, quantity float64, tokenAsset
interface{}) (interface{}, error)
```

Parameters:

- `to_account_id: string` – The account ID to receive the tokens.
- `quantity: number` – The total number of tokens to transfer.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message. The return value is the same as the TransferTokens method.

Return Value Example:

```
{     "msg":"Successfully transferred 50 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85
f0376152df (Org-Id: Org1MSP,  User-Id: admin) to account id:
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (Org-Id: Org1MSP, User-Id: user1)"
}
```

**`BulkTransfer`**
This method is used to perform bulk transfer of tokens from the caller account to the accounts that are specified in the `flow` object. The caller of this method must have an account already created.

```
Ctx.Token.BulkTransfer(flow []map[string]interface{}, tokenAsset
interface{}) (interface{}, error)
```

Parameters:

- `flow: object[]` – An array of JSON objects specifying the receiver details and quantity. The transfer quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file. For example:

  ```
  [{
      "to_org_id": "Org1MSP",
      "to_user_id": "user1",
      "quantity": 10
  }, {
      "to_org_id": "Org1MSP",
      "to_user_id": "user2",
      "quantity": 10
  }]
  ```

- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message that includes the number of tokens transferred. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "from_account_id":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "msg": "Successfully transferred 2 tokens from Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (Org-Id: Org1MSP, User-Id: user1)",
    "sub_transactions": [
```

```
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471aaa1210
c706e"
        },
        {
            "amount": 1,
            "to_account_id":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f0376
152df"
        }
    ]
}
```

### Methods for Token Behavior Management - Holdable Behavior

**Hold**

This method creates a hold on behalf of the owner of the tokens with the `to_account_id` account. A notary account is specified, which is responsible to either complete or release the hold. When the hold is created, the specified token balance from the payer is put on hold. A held balance cannot be transferred until the hold is either completed or released. The caller of this method must have an account already created.

```
Ctx.Token.Hold(operation_id string, to_account_id string, notary_account_id
string, quantity float64, TimeToExpiration string, tokenAsset))
(interface{}, error)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `to_account_id: string` – The ID of the account to receive the tokens.
- `notary__account_id: string` – The ID of the notary account.
- `quantity: number` – The total number of tokens to put on hold.
- `time_to_expiration: date` – The duration until the hold expires. Specify `0` for a permanent hold. Otherwise use the RFC-3339 format. For example, `2021-06-02T12`.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
 "msg": "account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
```

```
52df (org_id : Org1MSP, user_id : user1) is successfully holding 10
tokens",
}
```

**ExecuteHold**

This method completes a hold on tokens, transferring the specified quantity of tokens previously on hold to the receiver. If the `quantity` value is less than the actual hold value, then the remaining amount is available again to the original owner of the tokens. This method can be called only by the `AccountOwner` ID with the `notary` role.

```
Ctx.Token.ExecuteHold(operation_id string, quantity float64,
tokenAsset interface{}) (interface{}, error)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `quantity: number` – The total number of tokens to put on hold.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"msg":"Account Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f3
68642f622f (org_id : Org1MSP, user_id : user1) has successfully
executed '1' tokens(digiCurr101) from the hold with Operation Id
'op1'"}
```

**ReleaseHold**

This method releases a hold on tokens. The transfer is not completed and all held tokens are available again to the original owner. This method can be called by the `Account Owner` ID with the `notary` role within the specified time limit or by the payer, payee, or notary after the specified time limit.

```
Ctx.Token.ReleaseHold(operation_id string, tokenAsset interface{})
(interface{}, error)
```

Parameters:

- `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{"msg":"Successfully released '3' tokens from Operation Id 'op2' to Account
Id
oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642f
622f (org_id : Org1MSP, user_id : user1)"}
```

**GetOnHoldIds**

This method returns a list of all the holding IDs for a specified user and token.

```
Ctx.Account.GetOnHoldIDs(account_id string) (map[string]interface{}, error)
```

Parameters:

- `token_id` – The ID of the token.
- `org_id` – The membership service provider (MSP) ID of the user in the current network organization.
- `user_id` – The user name or email ID of the user.

Returns:

- On success, a JSON object with the list of holding IDs. A holding ID is formed by concatenating the asset type (`ohold`), the token name, the token ID, and the operation ID.

Return Value Example:

```
{"holding_ids":["ohold~loyaltok123~t1~op1"],"msg":"Holding Ids are:
[ohold~loyaltok123~t1~op1]"}
```

**GetOnHoldDetailsWithOperationID**

This method returns the on-hold transaction details for a specified operation ID and token..

```
Ctx.Hold.GetOnHoldDetailsWithOperationID(token_id string, operation_id
string) (Hold, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID that identifies the hold operation. Typically this ID is passed by the client application.

Returns:

- The return value is the same as the GetOnHoldDetailsWithOperationId method.
- On success, a promise object that includes the on-hold transaction details for the specified operation ID and token. The hold object includes the following properties:

  - `holding_id` – The holding ID of the transaction.
  - `operation_id: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
  - `from_account_id` – The account ID of the current owner of the on-hold tokens.
  - `to_account_id` – The account ID of the receiver.
  - `notary_account_id` – The account ID of the notary.
  - `token_id: string` – The ID of the saved token.
  - `quantity` – The amount of tokens that are on hold for the holding ID.
  - `time_to_expiration` – The duration until the hold expires.
- On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "AssetType": "ohold",
    "HoldingId": "ohold~digicur~digiCurr101~op1",
    "OperationId": "op1",
    "TokenName": "digicur",
    "FromAccountId":
"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f
368642f622f",
    "ToAccountId":
"oaccount~digicur~38848e87296d67c8a90918f78cf55f9c9baab2cdc8c928535471a
aa1210c706e",
    "NotaryAccountId":
"oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d8
5f0376152df",
    "TokenId": "digiCurr101",
    "Quantity": 2,
    "TimeToExpiration": "0"
}
```

**GetOnHoldBalanceWithOperationID**

This method returns the on-hold balance for a specified operation ID and token..

```
Ctx.Hold.GetOnHoldBalanceWithOperationID(token_id string, operation_id
string) (map[string]interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `operation_id: string` – A unique ID that identifies the hold operation. Typically this ID is passed by the client application.

Returns:

- On success, the on-hold balance of the specified operation ID and token. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
    "holding_balance": 10,
    "msg": "Current Holding Balance of OperationId opr_121 for token
digiCurr101 is : 10"
}
```

#### Methods for Token Behavior Management - Burnable Behavior

**`Burn`**
This method deactivates, or burns, tokens from the transaction caller's account. The caller of this method must have an account and the burner role. The quantity must be within the decimal values specified by the `decimal` parameter of the `divisible` behavior in the specification file.

```
Ctx.Token.Burn(quantity float64 , tokenAsset interface{}) (interface{},
error)
```

Parameters:

- `quantity: number` – The total number of tokens to burn.
- `tokenAsset` – The `tokenAsset` argument contains the reference of the token data to operate on.

Returns:

- On success, a success message. On error, a non-nil error object containing an error message.

Return Value Example:

```
{
 "msg":"Successfully burned 10 tokens from account id:
oaccount~digicur~682bb71de419602af74e3f226345ae308445ca51010737900c1d85f03761
52df (Org-Id: Org1MSP, User-Id: admin)"
}
```

## Go Methods for Token Conversion

Blockchain App Builder automatically generates methods that you can use to convert fungible tokens that use the Token Taxonomy Framework standard.

The token conversion methods include the concept of the **exchange pool**. The exchange pool account is funded by other token accounts. When you mint tokens, you can specify that a percentage of the minted tokens are transferred to the exchange pool account.

- Token Conversion Process
- Automatically Generated Token Conversion Methods
- Token Conversion SDK Methods

### Token Conversion Process

A typical flow for converting tokens follows these steps:

1. Call the `InitializeExchangePoolUser` method to initialize the exchange pool user.
2. Call the `CreateExchangePoolAccounts` method to create exchange pool accounts. Create an exchange pool account for every type of fungible token that you want to convert from or convert to.
3. Call the `AddConversionRate` method to set the conversion rate for each pair of tokens that you want to convert between.
4. Fund the exchange pool token accounts in one of the following ways:
   - Transfer tokens to the exchange pool token accounts using the standard transfer methods.
   - Call the `MintWithFundingExchangePoolToken` method when minting tokens, which can transfer a percentage of minted tokens to an exchange pool account.
5. Call the `TokenConversion` method to convert between two fungible tokens. A single user can convert tokens between two of their token accounts, or a pair of users can directly convert tokens from one account to another.
6. The exchange pool user can view the exchange pool account balances and account transactions.
   - Call the `GetAccount` method to view the balances of each of the exchange pool token accounts.
   - Call the `GetAccountTransactionHistory` and `GetAccountTransactionHistoryWithFilters` methods to view account transactions for each of the exchange pool token accounts.

### Automatically Generated Token Conversion Methods

Blockchain App Builder automatically generates methods to convert between different types of fungible tokens. Controller methods must be public to be invokable. Public method names begin with an upper case character. Method names that begin with a lower case character are private.

**`InitializeExchangePoolUser`**
This method initializes the exchange pool user, which is a one-time activity. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) InitializeExchangePoolUser(org_id string, user_id
string) (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.InitializeExchangePoolUs
er", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.TokenConvertor.InitializeExchangePoolUser(org_id,
```

```
user_id)
}
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the exchange pool user.

Return Value Example:

```
{
    "AssetType": "oconversion",
    "ConvertorId":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "OrgId": "Org1MSP",
    "UserId": "exchangepooluser"
}
```

**`CreateExchangePoolAccounts`**

This method creates exchange pool token accounts for a given array of token IDs. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) CreateExchangePoolAccounts(token_ids []string)
(interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.InitializeExchangePoolUser",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    var tokens []interface{}
    for _, tokenId := range token_ids {
        token, err := t.getTokenObject(tokenId)
        if err != nil {
            return nil, fmt.Errorf("error in getting from_token asset
details. Error: %s", err)
        }
        tokens = append(tokens, token.Interface())
    }
    return t.Ctx.TokenConvertor.CreateExchangePoolAccounts(tokens)
}
```

Parameters:

- `token_ids: string []` – An array of token IDs. You can specify up to ten token IDs.

Returns:

- On success, a list of objects that includes details of the exchange pool accounts that were created.

Return Value Example:

```
[
    {
        "AccountId":
"oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f
034",
        "Status": "created",
        "TokenId": "USD"
    },
    {
        "AccountId":
"oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a1
3c0",
        "Status": "created",
        "TokenId": "INR"
    }
]
```

**`AddConversionRate`**

This method adds a conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddConversionRate(from_token_id string,
to_token_id string, token_conversion_rate float64) (interface{},
error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.AddConversionRate",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.TokenConvertor.AddConversionToken(from_token_id,
to_token_id, token_conversion_rate)
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: float64` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
    "ConversionRate": 10
}
```

**GetConversionRate**

This method gets the current conversion rate for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
func (t *Controller) GetConversionRate(from_token_id string, to_token_id
string) (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.GetConversionRate", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    conversionRateId, err :=
t.Ctx.TokenConversionRate.GetConversionRateId(from_token_id, to_token_id)
    if err != nil {
        return nil, fmt.Errorf("error in getting conversationRateId. Error:
%s", err)
    }
    return t.Ctx.TokenConversionRate.Get(conversionRateId)
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
```

```
    "ConversionRate": 10
}
```

**UpdateConversionRate**

This method updates the current conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) UpdateConversionRate(from_token_id string,
to_token_id string, token_conversion_rate float64) (interface{},
error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.UpdateConversionRate",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return
t.Ctx.TokenConvertor.UpdateTokenConversionRate(from_token_id,
to_token_id, token_conversion_rate)
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: float64` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the updated conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c38
3d5e18b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
    "ConversionRate": 15
}
```

**MintWithFundingExchangePool**

This method mints tokens in the caller's account based on the specified token ID and quantity. A percentage of tokens from the minted quantity is then transferred to the exchange pool token account.

```
func (t *Controller) MintWithFundingExchangePool(token_id string,
token_quantity float64, percentage_token_to_exchangePool float64)
(interface{}, error) {
    token, err := t.getTokenObject(token_id)
    if err != nil {
        return nil, fmt.Errorf("error in getting from_token asset details.
Error: %s", err)
    }
    return
t.Ctx.TokenConvertor.MintWithFundingExchangePool(token.Interface(),
token_quantity, percentage_token_to_exchangePool)
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `token_quantity: float64` – The total number of tokens to mint.
- `percentage_token_to_exchange_pool: float64` – The percentage of minted tokens to transfer to the exchange pool token account.

Returns:

- On success, a message that indicates that minting and funding the exchange pool were successful.

Return Value Example:

```
{
    "msg": "successfully minted 100 tokens to AccountId:
'oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1'
(OrgId: Org1MSP, User-Id: admin) and Successfully transfered 80 tokens to
exchange pool account with AccountId:
'oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0'
(OrgId: Org1MSP, UserId: exchangepooluser) "
}
```

**TokenConversion**

This method converts tokens from the caller's account to the account specified by the `to_token_id`, `to_org_id`, and `to_user_id` values. This method can be called by the `Token Admin` of the chaincode and by any token account owner. An exchange pool user cannot call this method.

```
func (t *Controller) TokenConversion(from_token_id string, to_token_id
string, to_org_id string, to_user_id string, token_quantity float64)
(interface{}, error) {
    auth, err :=
```

```
t.Ctx.Auth.CheckAuthorization("TokenConversion.TokenConversion",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    from_token, err := t.getTokenObject(from_token_id)
    if err != nil {
        return nil, fmt.Errorf("error in getting from_token asset
details. Error: %s", err)
    }
    to_token, err := t.getTokenObject(to_token_id)
    if err != nil {
        return nil, fmt.Errorf("error in getting to_token asset
details. error: %s", err)
    }
    return
t.Ctx.TokenConvertor.TokenConversion(from_token.Interface(),
to_token.Interface(), to_org_id, to_user_id, token_quantity)
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `to_org_id: string` – The membership service provider (MSP) ID of the user in the current organization to receive the tokens.
- `to_user_id: string` – The user name or email ID of the user to receive the tokens.
- `token_quantity: float64` – The total number of tokens to transfer.

Returns:

- On success, a message that indicates the token conversion was successful.

Return Value Example:

```
{
    "msg": "succesfully converted 5 of tokens with tokenId: [USD] from
AccountId:
'oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb' (OrgId: Org1MSP, UserId: admin) to 75 of tokens with tokenId:
[INR] to AccountId:
'oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628
f1e' ( OrgId: Org1MSP, UserId: user ) as per the conversion rate of 15"
}
```

**GetConversionHistory**

This method returns the token conversion history for a specified token account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, and by the token account owner.

```
func (t *Controller) GetConversionHistory(token_id string, org_id string,
user_id string) (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.GetConversionHistory",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
    if err != nil {
        return nil, fmt.Errorf("error in generating the accoint_id. Error:
%s", err)
    }
    return t.Ctx.Account.GetTokenConversionHistory(account_id, org_id,
user_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object with conversion history details.

Return Value Example:

```
[
    {
        "balance": 95,
        "conversion_rate": 15,
        "converted_amount": 75,
        "from_account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",
        "from_token_id": "USD",
        "onhold_balance": 0,
        "timestamp": "2022-12-01T00:54:33+05:30",
        "to_account_id":
"oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628f1e",
        "to_token_id": "INR",
        "transacted_amount": 5,
        "transaction_id":
```

```
"otransaction~e1f5c32ab3cdc17a51ff0edda6bcc71b5acec3320a69f68a4ae455ed4
16657fa",
        "transaction_type": "TOKEN_CONVERSION_DEBIT"
    }
]
```

**GetConversionRateHistory**

This method returns the token conversion rate history for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
func (t *Controller) GetConversionRateHistory(from_token_id string,
to_token_id string) (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.GetConversionRateHistory
", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    conversion_rate_id, err :=
t.Ctx.TokenConversionRate.GetConversionRateId(from_token_id,
to_token_id)
    if err != nil {
        return nil, fmt.Errorf("error in getting conversionRateId.
Error: %s", err)
    }
    return t.Ctx.TokenConversionRate.History(conversion_rate_id)
}
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from, for the purpose of calculating the conversion rate.
- `to_token_id: string` – The ID of the token to convert to, for the purpose of calculating the conversion rate.

Returns:

- On success, a JSON object with conversion rate history details.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-01T00:48:58+05:30",
    "TxId":
"d6c5332278d33beddbc48e535029af424fef2129bf49f4906f9b527e101d95f1",
    "Value": {
      "AssetType": "oconversionRate",
      "ConversionRate": 15,
```

```
      "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
      "FromTokenId": "USD",
      "ToTokenId": "INR"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-01T00:47:15+05:30",
    "TxId":
"e8796578351e948827d5dfe242ab4be59019ae67d69d3bfd6db255a268d57017",
    "Value": {
      "AssetType": "oconversionRate",
      "ConversionRate": 10,
      "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
      "FromTokenId": "USD",
      "ToTokenId": "INR"
    }
  }
]
```

**GetExchangePoolUser**

This method returns the `org_id` and `user_id` values for the exchange pool user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetExchangePoolUser() (interface{}, error) {
    auth, err :=
t.Ctx.Auth.CheckAuthorization("TokenConversion.GetExchangePoolUser", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.TokenConvertor.GetExchangePoolUser()
}
```

Parameters:

- none

Returns:

- On success, a message with information about the exchange pool user.

Return Value Example:

```
{
    "AssetType": "oconversion",
    "ConvertorId":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "OrgId": "Org1MSP",
```

```
    "UserId": "exchangepooluser"
}
```

### Token Conversion SDK Methods

**`InitializeExchangePoolUser`**
This method initializes the exchange pool user, which is a one-time activity. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.InitializeExchangePoolUser(org_id string, user_id
string) (interface{}, error)
```

Parameters:

- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the exchange pool user.

Return Value Example:

```
{
    "AssetType": "oconversion",
    "ConvertorId":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "OrgId": "Org1MSP",
    "UserId": "exchangepooluser"
}
```

**`CreateExchangePoolAccounts`**
This method creates exchange pool token accounts for a given array of token IDs. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.CreateExchangePoolAccounts(tokens []interface{})
(interface{}, error)
```

Parameters:

- `token_ids: string []` – An array of token IDs.

Returns:

- On success, a list of objects that includes details of the exchange pool accounts that were created.

Return Value Example:

```
[
    {
```

```
        "AccountId":
"oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f034",
        "Status": "created",
        "TokenId": "USD"
    },
    {
        "AccountId":
"oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0",
        "Status": "created",
        "TokenId": "INR"
    }
]
```

**AddConversionToken**

This method adds tokens with a new conversion rate for a specified token. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.AddConversionToken(from_token_id string, to_token_id
string, token_conversion_rate float64) (interface{}, error)
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: float64` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
    "ConversionRate": 10
}
```

**Get**

This method gets the current conversion rate for a pair of tokens. This method can be called by the `Token Admin` of the chaincode and by any token account owner.

```
Ctx.TokenConversionRate.Get(id string) (ConversionRate, error)
```

Parameters:

- `id: string` – The ID of the token conversion rate object.

Returns:

- On success, a JSON representation of the conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c38
3d5e18b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
    "ConversionRate": 10
}
```

**`UpdateTokenConversionRate`**
This method updates the current conversion rate for a pair of tokens. The token conversion rate can be specified up to eight decimal places. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.UpdateTokenConversionRate(from_token_id string,
to_token_id string, token_conversion_rate float64) (interface{}, error)
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `token_conversion_rate: float64` – The rate at which to convert `from_token_id` token to the `to_token_id` token.

Returns:

- On success, a JSON representation of the updated conversion rate object.

Return Value Example:

```
{
    "AssetType": "oconversionRate",
    "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c38
3d5e18b150",
    "FromTokenId": "USD",
    "ToTokenId": "INR",
    "ConversionRate": 15
}
```

**MintWithFundingExchangePool**

This method mints tokens in the caller's account based on the specified token ID and quantity. A percentage of tokens from the minted quantity is then transferred to the exchange pool token account.

```
Ctx.TokenConvertor.MintWithFundingExchangePool(token interface{},
token_quantity float64, percentage_token_to_exchangePool float64)
(interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `token_quantity: float64` – The total number of tokens to mint.
- `percentage_token_to_exchange_pool: float64` – The percentage of minted tokens to transfer to the exchange pool token account.

Returns:

- On success, a message that indicates that minting and funding the exchange pool were successful.

Return Value Example:

```
{
    "msg": "successfully minted 100 tokens to AccountId:
'oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1'
(OrgId: Org1MSP, User-Id: admin) and Successfully transfered 80 tokens to
exchange pool account with AccountId:
'oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0'
(OrgId: Org1MSP, UserId: exchangepooluser) "
}
```

**TokenConversion**

This method converts tokens from the caller's account to the account specified by the `to_token_id`, `to_org_id`, and `to_user_id` values. This method can be called by the `Token Admin` of the chaincode and by any token account owner. An exchange pool user cannot call this method.

```
Ctx.TokenConvertor.TokenConversion(from_token interface{}, to_token
interface{}, to_org_id string, to_user_id string, token_quantity float64)
(interface{}, error)
```

Parameters:

- `from_token_id: string` – The ID of the token to convert from.
- `to_token_id: string` – The ID of the token to convert to.
- `to_org_id: string` – The membership service provider (MSP) ID of the user in the current organization to receive the tokens.
- `to_user_id: string` – The user name or email ID of the user to receive the tokens.

Returns:

- On success, a message that indicates the token conversion was successful.

Return Value Example:

```
{
    "msg": "succesfully converted 5 of tokens with tokenId: [USD] from
AccountId:
'oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb' (OrgId: Org1MSP, UserId: admin) to 75 of tokens with tokenId:
[INR] to AccountId:
'oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628
f1e' ( OrgId: Org1MSP, UserId: user ) as per the conversion rate of 15"
}
```

**GetTokenConversionHistory**

This method returns the token conversion history for a specified token account. This method can be called by the `Token Admin` of the chaincode and by the token account owner.

```
Ctx.Account.GetTokenConversionHistory(account_id string, org_id
string, user_id string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the fungible token account.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object with conversion history details.

Return Value Example:

```
[
    {
        "balance": 95,
        "conversion_rate": 15,
        "converted_amount": 75,
        "from_account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf4
8eb",
        "from_token_id": "USD",
        "onhold_balance": 0,
        "timestamp": "2022-12-01T00:54:33+05:30",
        "to_account_id":
"oaccount~25e2e66718b6dbb59aea9c32acebec60e09d912b2578d4933d377ae5d0628
f1e",
        "to_token_id": "INR",
```

```
        "transacted_amount": 5,
        "transaction_id":
"otransaction~e1f5c32ab3cdc17a51ff0edda6bcc71b5acec3320a69f68a4ae455ed416657f
a",
        "transaction_type": "TOKEN_CONVERSION_DEBIT"
    }
]
```

**history**

This method returns the token conversion rate history for a pair of tokens. This method can be called by the `Token Admin` of the chaincode, any `Org Admin`, and by any token account owner.

```
Ctx.TokenConversionRate.History(conversion_rate_id string) (interface{},
error)
```

Parameters:

- `conversion_rate_id: string` – The ID of the conversion rate object.

Returns:

- On success, a JSON object with conversion rate history details.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-01T00:48:58+05:30",
    "TxId":
"d6c5332278d33beddbc48e535029af424fef2129bf49f4906f9b527e101d95f1",
    "Value": {
      "AssetType": "oconversionRate",
      "ConversionRate": 15,
      "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
      "FromTokenId": "USD",
      "ToTokenId": "INR"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-01T00:47:15+05:30",
    "TxId":
"e8796578351e948827d5dfe242ab4be59019ae67d69d3bfd6db255a268d57017",
    "Value": {
      "AssetType": "oconversionRate",
      "ConversionRate": 10,
      "ConvertorRateId":
"oconversionRate~79eacc670928bbc4c9ba4ebee135c8b4d6411af3110f8a9b782c383d5e18
b150",
```

```
      "FromTokenId": "USD",
      "ToTokenId": "INR"
    }
  }
]
```

**GetExchangePoolUser**
This method returns the `org_id` and `user_id` values for the exchange pool user. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.TokenConvertor.GetExchangePoolUser()
```

Parameters:

- none

Returns:

- On success, a message with information about the exchange pool user.

Return Value Example:

```
{
    "AssetType": "oconversion",
    "ConvertorId":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
    "OrgId": "Org1MSP",
    "UserId": "exchangepooluser"
}
```

## Go Methods for Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for fungible tokens that use the Token Taxonomy Framework standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, transferring, and holding tokens. Additionally, other users cannot transfer tokens to or hold tokens in a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.

- Automatically Generated Account Status Methods
- Account Status SDK Methods

### Automatically Generated Account Status Methods

Blockchain App Builder automatically generates methods to manage token account status. Controller methods must be public to be invokable. Public method names begin

with an upper case character. Method names that begin with a lower case character are private.

**`GetAccountStatus`**
This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the token account owner. This method also supports data migration for existing chaincode that is upgraded to a newer version.

```
func (t *Controller) GetAccountStatus(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
account_id of (Org-Id: %s, User-Id: %s)", org_id, user_id)
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("AccountStatus.Get",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      accountStatus, err := t.Ctx.AccountStatus.GetAccountStatus(account_id)
      if err != nil {
            return t.Ctx.AccountStatus.GetDefaultAccountStatus(account_id)
      }
      return accountStatus, nil
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode, an `Org Admin` of the specified organization, or by the token account owner.

```
func (t *Controller) GetAccountStatusHistory(token_id string, org_id
string, user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
account_id of (Org-Id: %s, User-Id: %s)", org_id, user_id)
      }
      _, err = t.Ctx.Account.GetAccount(account_id)
      if err != nil {
            return nil, fmt.Errorf("error in GetAccountStatusHistory:
%s", err)
      }
      auth, err := t.Ctx.Auth.CheckAuthorization("AccountStatus.Get",
"TOKEN", map[string]string{"account_id": account_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      status_id, err :=
t.Ctx.AccountStatus.GenerateAccountStatusId(account_id)
      if err != nil {
            return nil, err
      }
      account_status_history, err :=
t.Ctx.AccountStatus.History(status_id)
      if err != nil {
            return []map[string]interface{}{}, nil
      }
      return account_status_history, nil
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the account status history.

Return Value Example:

```
[
  {
```

```
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7"
    }
  }
]
```

**`ActivateAccount`**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization. Deleted accounts cannot be activated.

```
func (t *Controller) ActivateAccount(token_id string, org_id string, user_id
string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
account_id of (Org-Id: %s, User-Id: %s)", org_id, user_id)
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("AccountStatus.ActivateAccount", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
```

```
      return t.Ctx.Account.ActivateAccount(account_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "active"
}
```

**`SuspendAccount`**

This method suspends a token account. Suspended accounts can still complete read operations. Users with suspended accounts cannot send, receive, mint, or burn tokens. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
func (t *Controller) SuspendAccount(token_id string, org_id string,
user_id string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id,
org_id, user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
account_id of (Org-Id: %s, User-Id: %s)", org_id, user_id)
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("AccountStatus.SuspendAccount", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
```

```
      return t.Ctx.Account.SuspendAccount(account_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "suspended"
}
```

**`DeleteAccount`**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization. After an account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot be changed to any other state. To delete an account, the account balance and the on-hold balance must be zero.

```
func (t *Controller) DeleteAccount(token_id string, org_id string, user_id
string) (interface{}, error) {
      account_id, err := t.Ctx.Account.GenerateAccountId(token_id, org_id,
user_id)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
account_id of (Org-Id: %s, User-Id: %s)", org_id, user_id)
      }
      auth, err :=
t.Ctx.Auth.CheckAuthorization("AccountStatus.DeleteAccount", "TOKEN",
map[string]string{"org_id": org_id})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.Account.DeleteAccount(account_id)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `org_id: string` – The membership service provider (MSP) ID of the user in the current organization.
- `user_id: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "deleted"
}
```

### Account Status SDK Methods

**`GetDefaultAccountStatus`**
This method gets the current status of a token account, with the status as `active` for any account that does not have account status stored in the ledger (because the account was created prior to the account status functionality).

```
Ctx.AccountStatus.GetDefaultAccountStatus(account_id string)
(FungibleAccountStatus, error)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
```

```
    "Status": "active"
}
```

**GetAccountStatus**
This method gets the current status of the token account.

```
Ctx.AccountStatus.GetAccountStatus(account_id string)
(FungibleAccountStatus, error)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**SaveAccountStatus**
This method saves the status object (if a status object is not present) for the token account, and sets the status to the specified value.

```
Ctx.AccountStatus.SaveAccountStatus(account_id string, status string)
```

Parameters:

- `account_id: string` – The ID of the token account.
- `status: string` – The status to set for the specified account. Only three values are supported: `active`, `suspended`, or `deleted`.

Returns:

- On success, a JSON representation of the account status object.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
```

```
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**

This method gets the history of the account status.

```
Ctx.AccountStatus.History(status_id string) (interface{}, error)
```

Parameters:

- `status_id: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
```

```
  }
]
```

**ActivateAccount**
This method activates a token account.

```
Ctx.Account.ActivateAccount(account_id: string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**SuspendAccount**
This method suspends a token account.

```
Ctx.Account.SuspendAccount(account_id string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
```

```
    "Status": "suspended"
}
```

**DeleteAccount**
This method deletes a token account.

```
Ctx.Account.DeleteAccount(account_id string) (interface{}, error)
```

Parameters:

- `account_id: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "deleted"
}
```

# ERC-721

Blockchain App Builder supports an extended version of the ERC-721 standard to work with non-fungible tokens.

- Input Specification File for ERC-721
- Scaffolded TypeScript NFT Project for ERC-721
- Scaffolded Go NFT Project for ERC-721

## Input Specification File for ERC-721

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

You can define standard assets and token assets that are based on the ERC-721 standard in the same specification file. The following restrictions apply to a specification file that includes an ERC-721 token asset definition:

- You cannot define token assets based on more than one standard in the same specification file.
- You cannot define multiple non-fungible token assets in a single specification file.

Custom attribute values for non-fungible token assets can be updated by the token owner.

For information on including standard assets in the specification file, see Input Specification File.

The following sample specification file for an ERC-721 token asset is available in the Blockchain App Builder package:

- `NFTArtCollectionMarketPlace with ERC-721-Typescript.yml`

In addition to the standard properties and sections, non-fungible token assets support the `behavior`, `anatomy` and `metadata` sections in the specification file. Non-fungible token assets also support the `standard` and `symbol` properties. The following example shows the structure of a specification file for an ERC-721 token asset:

```
assets:
    - name: ArtCollection #Asset name
      type: token #Asset type
      symbol: ART # Token symbol
      standard: erc721+ # Token standard

      anatomy:
          type: nonfungible # Token type
          unit: whole #Token unit

      behavior:
        - indivisible
        - singleton
        - mintable:
            max_mint_quantity: 20000
        - transferable
        - burnable
        - lockable
        - roles:
            minter_role_name: minter

      properties: # Custom asset attributes for non-fungible tokens

          - name: price # Custom asset attribute to set the price of a non-
fungible token in the marketplace
            type: number

          - name: on_sale_flag # Custom asset attribute maintains non-
fungible token selling status in the marketplace
            type: boolean

      metadata: # Use this section to maintain the metadata on the
blockchain. Only the user creating the non-fungible token can assign
metadata attribute values, which cannot be updated later.

          - name: painting_name
            type: string

          - name: description
            type: string
```

```
  - name: image
    type: string

  - name: painter_name
    type: string
```

**Table 7-6 Parameter Descriptions and Examples for an ERC-721 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`type:`** | You must specify `type: token` in the `assets` section. | `assets:`<br>`  - name: ArtCollection #Asset name`<br>`  type: token #Asset type` |
| **`symbol:`** | The `symbol` property represents the symbol that identifies the token contract, for example `ART`. Typically the symbol is 3 to 4 characters long. | `symbol: ART # Token symbol` |
| **`standard:`** | The `standard` property is mandatory for non-fungible tokens. It represents the token standard to follow during chaincode generation. In Blockchain App Builder, non-fungible tokens follow a partial version of the ERC-721 standard. For more information, see Limitations. | `standard: erc721+ # Token standard` |
| **`anatomy:`** | The `anatomy` section has two mandatory parameters for non-fungible tokens:<br>• `type: nonfungible` A non-fungible token is unique.<br>• `unit: whole` A whole token cannot be subdivided into smaller fractional units. | `anatomy:`<br>`  type: nonfungible # Token type`<br>`  unit: whole #Token unit` |

**Table 7-6 (Cont.) Parameter Descriptions and Examples for an ERC-721 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`behavior:`** | This section describes the capabilities and restrictions of the token. The `mintable`, `transferable`, `singleton`, and `indivisible` behaviors are mandatory for non-fungible tokens.<br>• `singleton`: This mandatory behavior supports a restriction so that there can be only one whole token in the class, which cannot be divided.<br>• `indivisible`: This mandatory behavior supports a restriction so that whole tokens cannot be subdivided into fractions.<br>• `mintable`: This mandatory behavior supports minting new token instances. The optional `max_mint_quantity` parameter specifies the total number of tokens that can be minted. If you do not specify the `max_mint_quantity` parameter, any number of tokens can be minted.<br>• `transferable`: This mandatory behavior | `behavior:`<br>`- indivisible`<br>`- singleton`<br>`- mintable:`<br>`max_mint_quantity: 20000`<br>`- transferable`<br>`- burnable`<br>`- lockable`<br>`- roles:`<br>`minter_role_name: minter` |

**Table 7-6 (Cont.) Parameter Descriptions and Examples for an ERC-721 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | supports transferring ownership of tokens.<br>• `burnable`: This optional behavior supports deactivating, or burning, tokens. Burning does not delete a token but instead places it in a permanent state where it cannot be used. Burning is not reversible.<br>• `lockable`: This behavior is optional and is supported only by non-fungible tokens. This behavior allows the token owner to lock a non-fungible token. A locked token cannot be transferred to or burned by any other users.<br>• `roles`: This optional behavior restricts token behaviors to users with specific roles. Currently two roles are available: `minter_role_name` and `burner_role_name`. If you do not specify roles, then any user can act as a minter or burner. For example, if the burner role is not specified, any account user implicitly has the | |

**Table 7-6 (Cont.) Parameter Descriptions and Examples for an ERC-721 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | burner role. If the burner role is specified, then during the token setup process, the `Token Admin` user must assign the burner role to other users explicitly. | |
| **`metadata:`** | The `metadata` property is optional and is supported only by non-fungible tokens. This property specifies metadata information, which is stored on the blockchain, for a non-fungible token. Metadata attribute values can be assigned only by the token owner who mints the token, and cannot be updated after the token is minted.<br>In the example, `name` is the name of the metadata attribute and `type` is the type of value that the attribute has. | `metadata:`<br>`painting_name`<br>`- name:`<br>`type: string`<br>`description`<br>`- name:`<br>`type: string`<br>`- name: image`<br>`type: string`<br>`painter_name`<br>`- name:`<br>`type: string` |

### Limitations

Blockchain App Builder provides partial support for the ERC-721 standard. Currently, the following ERC-721 events and methods are not supported.

Events:

- `event Transfer`
- `event Approval`
- `event ApprovalForAll`

Methods:

- `approve`
- `getApproved`
- `setApprovalForAll`

- `isApprovedForAll`

## Scaffolded TypeScript NFT Project for ERC-721

Blockchain App Builder takes the input from your NFT specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates NFT lifecycle classes and functions, including CRUD and non-CRUD methods, as well as a tokenization SDK. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to NFTs, see Scaffolded TypeScript Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated NFT Methods
  - Custom Methods
- NFT SDK Methods

### Model

Every tokenized model class extends the `OchainModel` class. Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class.

```
import * as yup from 'yup';
import { Id, Mandatory, Validate, Default, Embedded, Derived,
ReadOnly } from '../../lib/decorators';
import { OchainModel } from '../../lib/ochain-model';
import { STRATEGY } from '../../lib/utils';
import { EmbeddedModel } from '../../lib/ochain-embedded-model';

export class ArtCollectionMetadata extends
EmbeddedModel<ArtCollectionMetadata> {
    @Validate(yup.string())
    public painting_name: string;

    @Validate(yup.string())
    public description: string;

    @Validate(yup.string())
    public image: string;

    @Validate(yup.string())
    public painter_name: string;

}

@Id('tokenId')
export class ArtCollection extends OchainModel<ArtCollection> {
```

```
    public readonly assetType = 'otoken';

    @Mandatory()
    @Validate(yup.string().required().matches(/^[A-Za-z0-9][A-Za-
z0-9_-]*$/).max(16))
    public tokenId: string;

    @ReadOnly('artcollection')
    public tokenName: string;

    @Validate(yup.string().trim().max(256))
    public tokenDesc: string;

    @ReadOnly('ART')
    public symbol: string;

    @ReadOnly('erc721+')
    public tokenStandard: string;

    @ReadOnly('nonfungible')
    public tokenType: string;

    @ReadOnly('whole')
    public tokenUnit: string;


@ReadOnly(["indivisible","singleton","mintable","transferable","burnable","ro
les"])
    public behaviors: string[];

    @ReadOnly({minter_role_name: "minter"})
    public roles: object;

    @ReadOnly({max_mint_quantity: 20000})
    public mintable: object;

    @Validate(yup.string())
    public owner: string;

    @Validate(yup.string())
    public createdBy: string;

    @Validate(yup.string())
    public transferredBy: string;

    @Validate(yup.string())
    public creationDate: string;

    @Validate(yup.string())
    public transferredDate: string;

    @Validate(yup.bool())
    public isBurned: boolean;

    @Validate(yup.string())
```

```
    public burnedBy: string;

    @Validate(yup.string())
    public burnedDate: string;

    @Validate(yup.string().max(2000))
    public tokenUri: string;

    @Embedded(ArtCollectionMetadata)
    public metadata: ArtCollectionMetadata;

    @Validate(yup.number())
    public price: number;

    @Validate(yup.boolean())
    public on_sale_flag: boolean;

}
```

### Controller

The main controller class extends the `OchainController` class. There is only one main controller.

```
export class DigiCurrCCController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

### Automatically Generated NFT Methods

Blockchain App Builder automatically generates methods to support NFTs and NFT life cycles. You can use these methods to initialize NFTs, manage roles and accounts, and complete other NFT lifecycle tasks without any additional coding. Controller methods must have a `@Validator(...params)` decorator to be invokable.

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

**Methods for Access Control Management**

**addTokenAdmin**

This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async addTokenAdmin(orgId: string, userId: string) {
    await this.Ctx.ERC721Auth.checkAuthorization('ERC721ADMIN.addAdmin',
'TOKEN');
    return await this.Ctx.ERC721Admin.addAdmin(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully added Admin (orgId: Org1MSP, userId: User1)"}
```

**removeTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async removeTokenAdmin(orgId: string, userId: string) {
   await this.Ctx.ERC721Auth.checkAuthorization('ERC721ADMIN.removeAdmin',
'TOKEN');
   return await this.Ctx.ERC721Admin.removeAdmin(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg": "Successfully removed Admin (orgId: Org1MSP, userId: User1)"}
```

**isTokenAdmin**

This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`. A `Token Admin` can call this function on any other user in the blockchain network. Other users can call this method only on their own accounts.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async isTokenAdmin(orgId: string, userId: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ADMIN.isUserTokenAdmin',
'TOKEN');
    return await this.Ctx.ERC721Auth.isUserTokenAdmin(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

Return Value Example:

```
{"result": true}
```

**getAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by the `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async getAllTokenAdmins() {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ADMIN.getAllAdmins',
'TOKEN');
    return await this.Ctx.ERC721Admin.getAllAdmins();
}
```

Parameters:

- none

Returns:

- On success, an `admins` array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{"admins":[{"orgId":"Org1MSP","userId":"admin"}]}
```

### Methods for Token Configuration Management

**`init`**
This method is called when the chaincode is instantiated. Every `Token Admin` is identified by the `userId` and `orgId` information in the `adminList` parameter. The `userId` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `orgId` is the membership service provider (MSP) ID of the user in the current network organization. The `adminList` parameter is mandatory the first time you deploy the chaincode. If you are upgrading the chaincode, pass an empty list (`[]`). Any other information in the `adminList` parameter is ignored during upgrades.

```
@Validator(yup.array().of(yup.object()).nullable())
public async init(adminList: ERC721TokenAdminAsset[]) {
   await this.Ctx.ERC721Admin.initAdmin(adminList);
   await this.Ctx.ERC721Token.saveClassInfo(<NFT_NAME>);
   await this.Ctx.ERC721Token.saveDeleteTransactionInfo();
   return;
}
```

Parameters:

- `adminList array` – An array of `{orgId, userId}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

**`getAllTokens`**
This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@GetMethod()
@Validator()
public async getAllTokens() {
    await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.getAllTokens',
'TOKEN');
    return await this.Ctx.ERC721Token.getAllTokens();
}
```

Parameters:

- none

Returns:

- A list of all token assets in JSON format.

Return Value Example:


```
[
    {
        "key": "monalisa",
        "valueJson": {
            "metadata": {
                "PaintingName": "Mona_Lisa",
                "Description": "Mona Lisa Painting",
                "Image": "monalisa.jpeg",
                "PainterName": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa",
            "tokenName": "ravinft",
            "tokenDesc": "token Description",
            "symbol": "PNT",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "createdBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "creationDate": "2022-04-07T21:17:48.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
            "NftBasePrice": 100
        }
    },
    {
        "key": "monalisa1",
        "valueJson": {
            "metadata": {
```

```
                "PaintingName": "Mona_Lisa",
                "Description": "Mona Lisa Painting",
                "Image": "monalisa.jpeg",
                "PainterName": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa1",
            "tokenName": "ravinft",
            "tokenDesc": "token Description",
            "symbol": "PNT",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "createdBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "creationDate": "2022-04-07T21:17:59.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
            "NftBasePrice": 100
        }
    }
]
```

**getAllTokensByUser**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAllTokensByUser(orgId: string, userId: string) {
    const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
```

```
userId);
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.getAllTokensByUser'
, 'TOKEN', { accountId });
    return await this.Ctx.ERC721Token.getAllTokensByUser(accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- A list of token assets in JSON format.

Return Value Example:

```
[
    {
        "key": "monalisa",
        "valueJson": {
            "metadata": {
                "PaintingName": "Mona_Lisa",
                "Description": "Mona Lisa Painting",
                "Image": "monalisa.jpeg",
                "PainterName": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa",
            "tokenName": "ravinft",
            "tokenDesc": "token Description",
            "symbol": "PNT",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
```

```
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "createdBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "creationDate": "2022-04-07T21:17:48.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
            "NftBasePrice": 100
        }
    },
    {
        "key": "monalisa1",
        "valueJson": {
            "metadata": {
                "PaintingName": "Mona_Lisa",
                "Description": "Mona Lisa Painting",
                "Image": "monalisa.jpeg",
                "PainterName": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa1",
            "tokenName": "ravinft",
            "tokenDesc": "token Description",
            "symbol": "PNT",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "createdBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "creationDate": "2022-04-07T21:17:59.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
            "NftBasePrice": 100
```

```
        }
    }
]
```

**getTokenById**

This method returns a token object if the token is present in the state database. This method can be called only by a `Token Admin` of the chaincode or the token owner.

```
@GetMethod()
@Validator(yup.string())
public async getTokenById(tokenId: string) {
   await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.get',
'TOKEN', { tokenId });
   let token = await this.getTokenObject(tokenId);
   return token;
}
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Returns:

- The token asset in JSON format.

Return Value Example:

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
    "tokenName": "artcollection",
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
```

```
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "transferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "transferredDate": "2022-04-05T09:28:30.000Z",
    "isBurned": false,
    "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
    "price": 100,
    "on_sale_flag": true
}
```

**getTokenHistory**

This method returns the history for a specified token ID. This is an asynchronous method. This method can only be called when connected to the remote Oracle Blockchain Platform network. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async getTokenHistory(tokenId: string) {
   // await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.history',
'TOKEN');
   return await this.Ctx.ERC721Token.history(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
[
    {
        "trxId":
"ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e7",
        "timeStamp": 1649150910,
        "value": {
            "metadata": {
                "painting_name": "Mona_Lisa",
                "description": "Mona Lisa Painting",
                "image": "monalisa.jpeg",
                "painter_name": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa",
```

```
            "tokenName": "artcollection",
            "tokenDesc": "token description",
            "symbol": "ART",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "transferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "creationDate": "2022-04-05T08:30:42.000Z",
            "transferredDate": "2022-04-05T09:28:30.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
            "price": 100,
            "on_sale_flag": true
        }
    },
    {
        "trxId":
"cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9cb",
        "timeStamp": 1649149545,
        "value": {
            "metadata": {
                "painting_name": "Mona_Lisa",
                "description": "Mona Lisa Painting",
                "image": "monalisa.jpeg",
                "painter_name": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa",
            "tokenName": "artcollection",
```

```
                "tokenDesc": "token description",
                "symbol": "ART",
                "tokenStandard": "erc721+",
                "tokenType": "nonfungible",
                "tokenUnit": "whole",
                "behaviors": [
                    "indivisible",
                    "singleton",
                    "mintable",
                    "transferable",
                    "burnable",
                    "roles"
                ],
                "roles": {
                    "minter_role_name": "minter"
                },
                "mintable": {
                    "max_mint_quantity": 20000
                },
                "owner":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
                "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "creationDate": "2022-04-05T08:30:42.000Z",
                "transferredDate": "2022-04-05T09:05:45.000Z",
                "isBurned": false,
                "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
                "price": 100,
                "on_sale_flag": true
            }
        },
        {
            "trxId":
"702e61cc8d6d2982521023d0d5f3195900f35e146d6a90ef66daae551e6075d2",
            "timeStamp": 1649147729,
            "value": {
                "metadata": {
                    "painting_name": "Mona_Lisa",
                    "description": "Mona Lisa Painting",
                    "image": "monalisa.jpeg",
                    "painter_name": "Leonardo_da_Vinci"
                },
                "assetType": "otoken",
                "tokenId": "monalisa",
                "tokenName": "artcollection",
                "tokenDesc": "token description",
                "symbol": "ART",
                "tokenStandard": "erc721+",
                "tokenType": "nonfungible",
```

```
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "creationDate": "2022-04-05T08:30:42.000Z",
            "isBurned": false,
            "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
            "price": 100,
            "on_sale_flag": true
        }
    },
    {
        "trxId":
"e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c",
        "timeStamp": 1649147442,
        "value": {
            "metadata": {
                "painting_name": "Mona_Lisa",
                "description": "Mona Lisa Painting",
                "image": "monalisa.jpeg",
                "painter_name": "Leonardo_da_Vinci"
            },
            "assetType": "otoken",
            "tokenId": "monalisa",
            "tokenName": "artcollection",
            "tokenDesc": "token description",
            "symbol": "ART",
            "tokenStandard": "erc721+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
```

```
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "creationDate": "2022-04-05T08:30:42.000Z",
            "isBurned": false,
            "tokenUri": "\"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg\"",
            "price": 100,
            "on_sale_flag": false
        }
    }
]
```

**getTokenObject**

This is a utility method that returns an instance of the token for a specified token ID. This method is used by many of the automatically generated methods to fetch token objects. You can call this method as needed from your custom methods. When you create a tokenized asset or class, update the switch case with the corresponding `Token` class to return the correct token object. The `ochain sync` command in Blockchain App Builder automatically creates a switch case when a tokenized asset is created in the specification file. This method has no `@Validator()` method decorator, which means this method is not directly invokable and can only be called from other methods.

```
public async getTokenObject<T extends OchainModel<any>>(tokenId: string):
Promise<T> {
   if (!tokenId) {
        throw Error('TokenID cannot be null/empty.');
    }
    const token = await this.Ctx.ERC721Token.get(tokenId);
    if (token.tokenName && token.assetType && token.assetType === 'otoken') {
        let tokenAsset;
        switch (token.tokenName) {
           case '<NFT_NAME in lowercase>':
               tokenAsset = new <NFT_NAME>(token, false, true);
               return tokenAsset;
           default:
               throw new Error(`No token exists with ID [${tokenId}]`);
        }
    } else {
        throw new Error(`No token exists with ID [${tokenId}]`);
```

```
    }
}
```

Parameters:

- `tokenId: string` – The ID of the token.

**`ownerOf`**
This method returns the account ID of the owner of the specified token ID. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async ownerOf(tokenId: string) {
   return await this.Ctx.ERC721Token.ownerOf(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- A JSON object of the owner's account ID.

Return Value Example:

```
{"owner":
"oaccount~d6d22c3167e3c6ab9ee5653e1a008c37c20cc47ebb0229ca0aedfafe64c67
5b8"}
```

**`name`**
This method returns the name of the token class. Anyone can call this method.

```
@GetMethod()
@Validator()
public async name() {
   return await this.Ctx.ERC721Token.name();
}
```

Parameters:

- none

Returns:

- A JSON object of the token name.

Return Value Example:

```
{"tokenName": "artcollection"}
```

**symbol**
This method returns the symbol of the token class. Anyone can call this method.

```
@GetMethod()
@Validator()
public async symbol() {
   return await this.Ctx.ERC721Token.symbol();
}
```

Parameters:

- none

Returns:

- A JSON object of the token symbol.

Return Value Example:

```
{"symbol": "PNT"}
```

**tokenURI**
This method returns the URI of a specified token. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async tokenURI(tokenId: string) {
   return await this.Ctx.ERC721Token.tokenURI(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object of the token URI.

Return Value Example:

```
{"tokenURI": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
.ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"}
```

**totalSupply**
This method returns the total number of minted tokens. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async totalSupply() {
  await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.totalSupply',
```

```
'TOKEN');
  return await this.Ctx.ERC721Token.totalSupply();
}
```

Parameters:

- none

Returns:

- On success, a JSON object of the token count.

Return Value Example:

```
{"totalSupply": 3}
```

**totalNetSupply**
This method returns the total number of minted tokens minus the number of burned tokens. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async totalNetSupply() {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.totalNetSupply',
'TOKEN');
    return await this.Ctx.ERC721Token.getTotalMintedTokens();
}
```

Parameters:

- none

Returns:

- On success, a JSON object of the token count.

Return Value Example:

```
{"totalNetSupply": 1}
```

### Methods for Account Management

**createAccount**
This method creates an account for a specified user and token. An account must be created for any user who will have tokens at any point. Accounts track the number of NFTs a user has. Users must have accounts in the network to complete token-related operations. You can create only one NFT account per user.
An account ID is an alphanumeric set of characters, prefixed with `oaccount~` and followed by an SHA-256 hash of the membership service provider ID (`orgId`) of the user in the current network organization, the user name or email ID (`userId`) of the

instance owner or the user who is logged in to the instance, and the constant string `nft`. This method can be called only by the `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async createAccount(org_id: string, user_id: string, token_type:
string) {
  await this.Ctx.Auth.checkAuthorization("ACCOUNT.createAccount", "TOKEN",
{ org_id });
  return await this.Ctx.Account.createAccount(org_id, user_id, token_type);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenType: string` – The only supported token type is `nonfungible`.

Returns:

- On success, a JSON object of the account that was created. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:

```
{
    "assetType": "oaccount",
    "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "bapAccountVersion": 0,
    "userId": "admin",
    "orgId": "Org1MSP",
    "tokenType": "nonfungible",
    "noOfNfts": 0
}
```

**`balanceOf`**

This method returns the total number of NFTs that a specified user holds. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string())
 public async balanceOf(orgId: string, userId: string) {
     await this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.balanceOf',
'TOKEN', { orgId, userId });
     const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
     return await this.Ctx.ERC721Account.balanceOf(accountId);
 }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- A JSON object of the current NFT count.

Return Value Example:

```
{"totalNfts": 0}
```

**`getAllAccounts`**

This method returns a list of all accounts. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@GetMethod()
@Validator()
public async getAllAccounts() {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.getAllAccounts',
'TOKEN');
    return await this.Ctx.ERC721Account.getAllAccounts();
}
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
    {
        "key":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "valueJson": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
```

```
    }
]
```

**getAccountByUser**

This method returns account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAccountByUser(orgId: string, userId: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.getAccountByUser',
'TOKEN', { orgId, userId });
    return await this.Ctx.ERC721Account.getAccountWithStatusByUser(orgId,
userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `bapAccountVersion` – An account object parameter for internal use.
- `status` – The current status of the user account.
- `accountId` – The ID of the user account.
- `userId` – The user name or email ID of the user.
- `orgId` – The membership service provider (MSP) ID of the user in the current organization.
- `tokenType` – The type of token that the account holds.
- `noOfNfts` – The total number of NFTs held by the account.

Return Value Example:

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "accountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
  "userId": "idcqa",
  "orgId": "appdev",
  "tokenType": "nonfungible",
  "noOfNfts": 0
}
```

**getUserByAccountId**
This method returns the user details of a specified account. This method can be called by any user.

```
@GetMethod()
@Validator(yup.string())
public async getUserByAccountId(accountId: string) {
    return await this.Ctx.ERC721Account.getUserByAccountId(accountId);
}
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`orgId` and `userId`).

Return Value Example:

```
{
  "userId": "admin",
  "orgId": "Org1MSP"
}
```

**getAccountHistory**
This method returns account history for a specified user. This is an asynchronous method. This method can be called only by the `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAccountHistory(orgId: string, userId: string) {
    const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.history',
'TOKEN', { accountId });
    return await this.Ctx.ERC721Account.history(accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object of the account history. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example:


```
[
    {
        "trxId":
"6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4",
        "timeStamp": 1649151044,
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion" : 5,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3ecf53a3",
        "timeStamp": 1649151022,
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion" : 4,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 2
        }
    },
    {
        "trxId":
"ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e7",
        "timeStamp": 1649150910,
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion" : 3,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9cb",
        "timeStamp": 1649149545,
        "value": {
```

```
            "assetType": "oaccount",
            "bapAccountVersion" : 2,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 0
        }
    },
    {
        "trxId":
"e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c",
        "timeStamp": 1649147442,
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion" : 1,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"d2d1f9c898707ae831e9361bc25da6369eac37b10c87dc04d18d6f3808222f08",
        "timeStamp": 1649137534,
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion" : 0,
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 0
        }
    }
]
```


### Methods for Role Management

**addRole**

This method adds a role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async addRole(role: string, orgId: string, userId: string) {
   const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
   await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.addRoleMember',
'TOKEN');
   return await this.Ctx.ERC721Token.addRoleMember(role, accountId);
}
```

Parameters:

- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg": "Successfully added role 'minter' to Account Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin)"}
```

**removeRole**

This method removes a role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async removeRole(role: string, orgId: string, userId: string) {
    const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.removeRoleMember',
'TOKEN');
    return await this.Ctx.ERC721Token.removeRoleMember(role, accountId);
}
```

Parameters:

- `role: string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message with account details.

Return Value Example:

```
{"msg": "Successfully removed role 'minter' from Account Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729d
ba (Org-Id: Org1MSP, User-Id: user1)"}
```

**getAccountsByRole**

This method returns a list of all account IDs for a specified role. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getAccountsByRole(role: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ROLE.getAccountsByRole',
'TOKEN');
    return await this.Ctx.ERC721Role.getAccountsByRole(role);
}
```

Parameters:

- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{
    "accounts": [

"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d"
    ]
}
```

**getUsersByRole**

This method returns a list of all users for a specified role. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getUsersByRole(role: string) {
```

```
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ROLE.getUsersByRole', 'TOKEN');
    return await this.Ctx.ERC721Role.getUsersByRole(role);
}
```

Parameters:

- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of the user objects (`orgId` and `userId`).

Return Value Example:

```
{
    "users": [
        {
            "userId": "admin",
            "orgId": "Org1MSP"
        }
    ]
}
```

**isInRole**

This method returns a Boolean value to indicate if a user has a specified role. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string())
public async isInRole(orgId: string, userId: string, role: string) {
    const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
    await this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.isInRole',
'TOKEN',{ accountId });
    return { result: await this.Ctx.ERC721Token.isInRole(role, accountId) };
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON string of the Boolean result.

Return Value Example:

```
{"result":"true"}
```

### Methods for Transaction History Management

**`getAccountTransactionHistory`**
This method returns account transaction history for a specified user. This is an asynchronous method. This method can be called only by the `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAccountTransactionHistory(orgId: string, userId:
string) {
    const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.getAccountTransac
tionHistory', 'TOKEN', { accountId });
    return await
this.Ctx.ERC721Account.getAccountTransactionHistory(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
[
    {
        "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632
bdaf5d4",
        "timestamp": "2022-04-05T09:30:44.000Z",
        "tokenId": "monalisa1",
        "noOfNfts": 1,
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "transactionType": "BURN"
    },
    {
        "transactionId":
"otransaction~a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3
ecf53a3",
        "timestamp": "2022-04-05T09:30:22.000Z",
        "tokenId": "monalisa1",
```

```
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e
7",
        "timestamp": "2022-04-05T09:28:30.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "CREDIT"
    },
    {
        "transactionId":
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9c
b",
        "timestamp": "2022-04-05T09:05:45.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "DEBIT"
    },
    {
        "transactionId":
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337
c",
        "timestamp": "2022-04-05T08:30:42.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "MINT"
    }
]
```

**getAccountTransactionHistoryWithFilters**

This method returns account transaction history for a specified user, filtered by `PageSize`, `Bookmark`, `startTime` and `endTime`. This is an asynchronous method. This method can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by the `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.object().nullable())
public async getAccountTransactionHistoryWithFilters(orgId: string, userId:
string, filters ?: Filters) {
    const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721ACCOUNT.getAccountTransactionHi
storyWithFilters', 'TOKEN', { accountId });
    return await
```

```
this.Ctx.ERC721Account.getAccountTransactionHistoryWithFilters(accountI
d, filters)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `filters: object` – An object of the Filter class that contains four attributes: `pageSize`, `bookmark`, `startTime` and `endTime`.

Return Value Example:

```
[
    {
        "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632
bdaf5d4",
        "timestamp": "2022-04-05T09:30:44.000Z",
        "tokenId": "monalisa1",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "transactionType": "BURN"
    },
    {
        "transactionId":
"otransaction~a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3
ecf53a3",
        "timestamp": "2022-04-05T09:30:22.000Z",
        "tokenId": "monalisa1",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427
d9ed4e7",
        "timestamp": "2022-04-05T09:28:30.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba",
        "transactionType": "CREDIT"
    },
    {
        "transactionId":
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d63802
0c1f9cb",
```

```
        "timestamp": "2022-04-05T09:05:45.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "DEBIT"
    },
    {
        "transactionId":
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337
c",
        "timestamp": "2022-04-05T08:30:42.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "MINT"
    }
]
```

**getTransactionById**

This method returns transaction history for a specified transaction ID. This is an asynchronous method. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string())
public async getTransactionById(transactionId: string) {
    return await
this.Ctx.ERC721Transaction.getTransactionById(transactionId);
}
```

Parameters:

- `transactionId: string` – The id of the transaction, which is the prefix `otransaction~` followed by the 64-bit hash in hexadecimal format.

Return Value Example:

```
{
    "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d
4",
    "history": [
        {
            "trxId":
"6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4",
            "timeStamp": 1649151044,
            "value": {
                "assetType": "otransaction",
                "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d
4",
                "tokenId": "monalisa1",
```

```
                "fromAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
                "toAccountId": "",
                "triggeredByAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
                "transactionType": "BURN",
                "timestamp": "2022-04-05T09:30:44.000Z",
            }
        }
    ]
}
```

**deleteHistoricalTransactions**

This method deletes transactions older than a specified time stamp in the state database. This is an asynchronous method. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.date())
public async deleteHistoricalTransactions(timeToExpiration: Date) {
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721TRANSACTION.deleteTransac
tions', 'TOKEN');
    return await
this.Ctx.ERC721Transaction.deleteTransactions(timeToExpiration);
}
```

Parameters:

- `timestamp: string` – A time stamp. All transactions before the time stamp will be deleted.

Return Value Example:

```
{
    "msg": "Successfuly deleted transaction older than date: Thu Apr
07 2022 21:18:59 GMT+0000 (Coordinated Universal Time).",
    "transactions": [

"otransaction~30513757d8b647fffaafac440d743635f5c1b2e41b25ebd6b70b5bbf7
8a2643f",

"otransaction~ac0e908c735297941ba58bb208ee61ff4816a1e54c090d68024f82adf
743892b"
    ]
}
```

### Methods for Token Behavior Management - Mintable Behavior

**create<Token Name>Token**

This method creates (mints) an NFT. The asset and associated properties are saved in the state database. The caller of this transaction must have a token account. The

caller of this transaction becomes the owner of the NFT. If the token specification file includes the `roles` section for `behaviors` and the `minter_role_name` property for `roles`, then the caller of the transaction must have the minter role. Otherwise, any caller can mint NFTs.

```
@Validator(< Token Class >)
public async create< Token Name >Token(tokenAsset: <Token Class>) {
    return await this.Ctx.ERC721Token.createToken(tokenAsset);
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to mint. For more information about the properties of the token asset, see the input specification file.

Returns:

- On success, a JSON token asset object that includes the following properties:
- `metadata` – JSON information that describes the token.
- `createdBy` – The account ID of the user who called the transaction to mint the token.
- `creationDate` – The time stamp of the transaction.
- `isBurned` – A Boolean value that indicates if the NFT identified by `tokenId` is burned.
- `tokenName` – The name of the token.
- `tokenDesc` – The description of the token.
- `symbol` – The symbol of the token.
- `tokenStandard` – The standard of the token.
- `tokenType` – The type of token held by this account.
- `tokenUnit` – The unit of the token.
- `behaviors` – A description of all token behaviors.
- `mintable` – A description of the properties of mintable behavior. The `max_mint_quantity` property specifies the maximum number of NFTs of this token class that can be created.
- `owner` – The account ID of the current owner of the token. During the minting process, the caller of this method becomes the owner of the token.
- `tokenUri` – The URI of the token.

Return Value Example:

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
```

```
    "tokenName": "artcollection",
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "isBurned": false,
    "tokenUri": "\"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg\"",
    "price": 100,
    "on_sale_flag": false
}
```

**update<Token Name>Token**

This method updates token properties. This method can be called only by the user who is the owner or creator of the token. After a token asset is created, only the token owner can update the token custom properties. If the user is both token owner and creator of a token, they can also update the `TokenDesc` property. Token metadata cannot be updated. You must pass all token properties to this method, even if you want to update only certain properties.

```
@Validator(<Token Class>)
public async update<Token name>Token(tokenAsset: <Token Class>) {
    return await this.Ctx.ERC721Token.updateToken(tokenAsset);
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to update. For more information about the properties of the token asset, see the input specification file.

Returns:

- On success, an updated JSON token asset object

Return Value Example:

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
    "tokenName": "artcollection",
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "isBurned": false,
    "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
    "price": 100,
    "on_sale_flag": true
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`safeTransferFrom`**
This is an asynchronous function. This method transfers ownership of the specified NFT from the caller to another account. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.
- The caller of the function is the sender.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string(), yup.string().max(2000))
public async safeTransferFrom(fromOrgId: string, fromUserId: string,
toOrgId: string, toUserId: string, tokenId: string, data?: string) {
    const tokenAsset = await this.getTokenObject(tokenId);
    const fromAccountId = await
this.Ctx.ERC721Account.generateAccountId(fromOrgId, fromUserId);
    const toAccountId = await
this.Ctx.ERC721Account.generateAccountId(toOrgId, toUserId);
    return await this.Ctx.ERC721Token.safeTransferFrom(fromAccountId,
toAccountId, tokenAsset, data);
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender and token owner in the current organization.
- `fromUserId: string` – The user name or email ID of the sender and token owner.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenId: string` – The ID of the token to transfer.
- `data: string` – Optional additional information to store in the transaction record.

Returns:

- On success, a message with the sender and receiver account details.

Return Value Example:

```
{"msg": "Successfully transferred NFT token: 'monalisa' from Account-
Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729d
ba (Org-Id: Org1MSP, User-Id: user1)"}
```

**`transferFrom`**

This is an asynchronous function. This method transfers ownership of the specified NFT from a sender account to a receiver account. It is the responsibility of the caller to pass the correct parameters. This method can be called by any user, not only the token owner. This method includes the following validations:

- The token exists and is not burned.

- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string())
public async transferFrom(fromOrgId: string, fromUserId: string, toOrgId:
string, toUserId: string, tokenId: string) {
    const tokenAsset = await this.getTokenObject(tokenId);
    const fromAccountId = await
this.Ctx.ERC721Account.generateAccountId(fromOrgId, fromUserId);
    const toAccountId = await
this.Ctx.ERC721Account.generateAccountId(toOrgId, toUserId);
    return await this.Ctx.ERC721Token.transferFrom(fromAccountId,
toAccountId, tokenAsset);
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender in the current organization.
- `fromUserId: string` – The user name or email ID of the sender.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenId: string` – The ID of the token to transfer.

Returns:

- On success, a message with the sender and receiver account details.

Return Value Example:

```
{"msg": "Successfully transferred NFT token: 'monalisa' from Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1) to Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin)"}
```

### Methods for Token Behavior Management - Burnable Behavior

**`burn`**

This method deactivates, or burns, the specified NFT from the caller's account. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a

`burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) or owner of the token can burn the token.

```
@Validator(yup.string())
public async burn(tokenId: string) {
    const tokenAsset = await this.getTokenObject(tokenId);
    return await this.Ctx.ERC721Token.burn(tokenAsset);
}
```

Parameters:

- `tokenId: string` – The ID of the token to burn.

Returns:

- On success, a message with the account details.

Return Value Example:

```
{"msg": "Successfully burned NFT token: 'monalisa1' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin)"}
```

**burnNFT**

This method deactivates, or burns, the specified NFT from the caller's account, and returns a token object and token history. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a `burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) or owner of the token can burn the token.

```
@Validator(yup.string())
public async burnNFT(tokenId: string) {
    const token = await this.Ctx.ERC721Token.get(tokenId)
    if (token.isBurned === true) {
      throw new Error(`token with tokenId ${tokenId} is already
burned`);
    }
    const tokenHistory = await this.Ctx.ERC721Token.history(tokenId);
    await this.burn(tokenId);
    token.tokenId = parseInt(token.tokenId);
    if(Number.isNaN(token.tokenId)) {
      throw new Error(`tokenId is expected to be integer but found $
{tokenId}`)
    }
    token.isBurned = true;
    return {...token, tokenHistory: JSON.stringify(tokenHistory)};
}
```

Parameters:

- `tokenId: string` – The ID of the token to burn.

Returns:

- On success, a token object that includes token history information.

Return Value Example:


```
{
    "assetType": "otoken",
    "tokenId": 1,
    "tokenName": "artcollection",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2023-08-22T07:36:50.000Z",
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "isBurned": true,
    "tokenUri": "example.com",
    "price": 120,
    "on_sale_flag": false,
    "tokenHistory":
"[{\"trxId\":\"732438a85b5e8fc76c5254e54602b29d583543b103fafb5a28c0df384428bb
50\",\"timeStamp\":\"2023-08-22T07:36:50.000Z\",\"value\":
{\"assetType\":\"otoken\",\"tokenId\":\"1\",\"tokenName\":\"artcollection\",\
"symbol\":\"ART\",\"tokenStandard\":\"erc721+
\",\"tokenType\":\"nonfungible\",\"tokenUnit\":\"whole\",\"behaviors\":
[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"burnable\",\"r
oles\"],\"roles\":{\"minter_role_name\":\"minter\"},\"mintable\":
{\"max_mint_quantity\":20000},\"createdBy\":\"oaccount~42e89f4c72dfde95028148
76423c6da630d466e87436dd1aae201d347ad1288d\",\"creationDate\":\"2023-08-22T07
:36:50.000Z\",\"owner\":\"oaccount~42e89f4c72dfde9502814876423c6da630d466e874
36dd1aae201d347ad1288d\",\"isBurned\":false,\"tokenUri\":\"example.com\",\"pr
ice\":120,\"on_sale_flag\":false}}]"
}
```

### Custom Methods

You can use the token SDK methods to write custom methods for your business application.

The following example shows how to use token SDK methods in custom methods. When the `sell` method is called, it posts a token for sale for a specified price.

```
@Validator(yup.string(), yup.number())
public async sell(token_id: string, selling_price: number) {
    try {
        const token = await this.Ctx.ERC721Token.get(token_id);
        const t = new ArtCollection(token)
 /**  * price is a custom asset
      attribute to set the price of a non-fungible token in the
      marketplace  */
        t.price =  selling_price;
 /**  * on_sale_flag is a
      custom asset attribute that maintains non-fungible token selling
status in the
      marketplace  */
        t.on_sale_flag = true;
        await this.Ctx.ERC721Token.updateToken(t);
        let msg = `Token ID : '${token_id}' has been posted for
selling in the marketplace'`;
        return {msg}
        } catch(error) {
            throw new Error(error.message);
    }
}
```

### NFT SDK Methods

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

The NFT SDK provides an access control function. Some methods can be called only by a `Token Admin` or `Account Owner` of the token. You can use this feature to ensure that operations are carried out only by the intended users. Any unauthorized access

results in an error. To use the access control function, import the `Authorization` class from the `../lib/erc721-auth` module.

```
import { ERC721Authorization } from '../lib/erc721-auth';
```

**checkAuthorization**

Use this method to add an access control check to an operation. This is an asynchronous function. Most automatically generated methods include access control. Certain token methods can be run only by the `ERC721Admin` or `Account Owner` of the token or by the `MultipleAccountOwner` for users with multiple accounts. The `checkAuthorization` method is part of the `Authorization` class, which you access via the `Ctx` object. The access control mapping is described in the `../lib/constant.ts` file, as shown in the following text. You can modify access control by editing the `../lib/constant.ts` file. To use your own access control or to disable access control, remove the access control code from the automatically generated controller methods and custom methods.

```
export const TOKENACCESS = {
  ADMIN: {
    isUserTokenAdmin: ['Admin', 'MultipleAccountOwner'],
    addAdmin: ['Admin'],
    removeAdmin: ['Admin'],
    getAllAdmins: ['Admin'],
  },
  TOKEN: {
    save: ['Admin'],
    getAllTokens: ['Admin'],
    get: ['Admin'],
    update: ['Admin'],
    getDecimals: ['Admin'],
    getTokensByName: ['Admin'],
    addRoleMember: ['Admin'],
    removeRoleMember: ['Admin'],
    isInRole: ['Admin', 'AccountOwner'],
    getTotalMintedTokens: ['Admin'],
    getNetTokens: ['Admin'],
  },
  ROLE: {
    getAccountsByRole: ['Admin'],
    getUsersByRole: ['Admin'],
  },
  TRANSACTION: {
    deleteTransactions: ['Admin'],
  },
  ACCOUNT: {
    createAccount: ['Admin'],
    getAllAccounts: ['Admin'],
    getAccountsByUser: ['Admin', 'MultipleAccountOwner'],
    getAccount: ['Admin', 'AccountOwner'],
    history: ['Admin', 'AccountOwner'],
    getAccountTransactionHistory: ['Admin', 'AccountOwner'],
    getAccountBalance: ['Admin', 'AccountOwner'],
    getAccountOnHoldBalance: ['Admin', 'AccountOwner'],
```

```
    getOnHoldIds: ['Admin', 'AccountOwner'],
  },
  ERC721ADMIN: {
    isUserTokenAdmin: ['Admin'],
    addAdmin: ['Admin'],
    removeAdmin: ['Admin'],
    getAllAdmins: ['Admin'],
  },
  ERC721TOKEN: {
    getAllTokens: ['Admin'],
    getAllTokensByUser: ['Admin', 'AccountOwner'],
    get: ['Admin', TOKEN_OWNER],
    getTokensByName: ['Admin'],
    addRoleMember: ['Admin'],
    removeRoleMember: ['Admin'],
    isInRole: ['Admin', 'AccountOwner'],
    totalSupply: ['Admin'],
    totalNetSupply: ['Admin'],
    history: ['Admin'],
  },
  ERC721ROLE: {
    getAccountsByRole: ['Admin'],
    getUsersByRole: ['Admin'],
  },
  ERC721TRANSACTION: {
    deleteTransactions: ['Admin'],
  },
  ERC721ACCOUNT: {
    createAccount: ['Admin'],
    getAllAccounts: ['Admin'],
    getAccountsByUser: ['Admin', 'MultipleAccountOwner'],
    history: ['Admin', 'AccountOwner'],
    getAccountTransactionHistory: ['Admin', 'AccountOwner'],
    getAccountTransactionHistoryWithFilters: ['Admin', 'AccountOwner'],
    balanceOf: ['Admin', 'MultipleAccountOwner'],
  }
}
```

```
Ctx.ERC721Auth.checkAuthorization(classFuncName: string, ...args)
```

Parameters:

- `classFuncName: string` – The map value between the class and methods as described in the `../lib/constant.ts` file.
- `...args` – A variable argument where `args[0]` takes the constant `'TOKEN'` and `args[1]` takes the `accountId` parameter to add an access control check for an `AccountOwner`. To add an access control check for a `MultipleAccountOwner`, `args[1]` takes the `orgId` parameter and `args[2]` takes the `userId` parameter.

Returns:

- On success, a promise. On error, a rejection with an error message.

Examples:
`Admin` access

```
await this.Ctx.ERC721Auth.checkAuthorization('ADMIN.addAdmin', 'TOKEN');
```

`AccountOwner` access

```
await this.Ctx.ERC721Auth.checkAuthorization('ACCOUNT.getAccountBalance',
'TOKEN', accountId);
```

`MultipleAccountOwner` access

```
await this.Ctx.ERC721Auth.checkAuthorization('ADMIN.isUserTokenAdmin',
'TOKEN', orgId, userId);
```

**isUserTokenAdmin**
This method returns the Boolean value `true` if the caller of the function is a `Token Admin`. Otherwise the method returns `false`. This is an asynchronous static function.

```
Ctx.ERC721Auth.isUserTokenAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user in the current network organization.
- `userId` – The user name or email ID of the user.

Returns:

- A Boolean response and an error message if an error is encountered.

Example:

```
await this.Ctx.Auth.isUserTokenAdmin('Org1MSP', 'user1');
```

```
{"result":false}
```

**addAdmin**
This method adds a user as a `Token Admin` of the token chaincode.

```
Ctx.ERC721Admin.addAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user added as a `Token Admin` of the token chaincode. On error, a non-nil error object that contains an error message.

Example:
```
await this.Ctx.ERC721Admin.addAdmin(orgId, userId)

{"msg": "Successfully added Admin (orgId: Org1MSP, userId: user1)"}
```

**`removeAdmin`**
This method removes a user as a `Token Admin` of the token chaincode.

```
Ctx.ERC721Admin.removeAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user removed as a `Token Admin` of the token chaincode. On error, a non-nil error object that contains an error message.

Example:
```
await this.Ctx.ERC721Admin.RemoveAdmin(orgId, userId)

{"msg": "Successfully removed Admin (orgId: Org1MSP, userId: user1)"}
```

**`getAllAdmins`**
This method returns a list of all `Token Admin` users.

```
Ctx.ERC721Admin.getAllAdmins()
```

Parameters:

- none

Returns:

- On success, a list of all `Token Admin` users. On error, a non-nil error object that contains an error message.

Example:
```
await this.Ctx.ERC721Admin.getAllAdmins()

{
    "admins": [
        {
```

```
            "orgId": "Org1MSP",
            "userId": "admin"
        }
    ]
}
```

### Methods for Token Configuration Management

The token configuration management methods are based on the ERC-721 standard. To use the token configuration management methods, import the `Token` class from the `../lib/erc721-token` module.

**`totalSupply`**
This method returns the total number of minted NFTs. This is an asynchronous function.

```
Ctx.ERC721Token.totalSupply()
```

Parameters:

- none

Returns:

- On success, the total net tokens, in the number data type. On error, it returns with an error message.

Example:

```
await this.Ctx.ERC721Token.totalSupply(tokenAsset);
```

```
2000
```

**`get`**
This method returns the specified token object if it is present in the state database. This is an asynchronous static function.

```
Ctx.ERC721Token.get(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a promise that includes a JSON object of the token asset. On error, a rejection with an error message

Example:

```
await this.Ctx.ERC721Token.get(tokenId);
```

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
```

```
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
    "tokenName": "artcollection",
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "transferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "transferredDate": "2022-04-05T09:28:30.000Z",
    "isBurned": false,
    "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
    "price": 100,
    "on_sale_flag": true
}
```

**isTokenType**

This method indicates whether a token asset exists with the specified ID. This is an asynchronous static function.

```
Ctx.ERC721Token.isTokenType(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a promise with `true` if a token asset exists with the specified ID. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Token.isTokenType(tokenId);
```

```
true
```

**`createToken`**

This method creates a token and saves its properties in the state database. This method can be called only by users with the minter role. This is an asynchronous function.

```
Ctx.ERC721Token.createToken(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to create.

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Token.createToken(tokenAsset);
```

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
    "tokenName": "artcollection",
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
```

```
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "isBurned": false,
    "tokenUri": "\"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg\"",
    "price": 100,
    "on_sale_flag": false
}
```

**updateToken**

This method updates token properties. This method can be called only by the owner or creator of the token. After a token asset is created, only the token owner can update the token custom properties. If the user is both token owner and creator of a token, they can also update the `TokenDesc` property. Token metadata cannot be updated. You must pass all token properties to this method, even if you want to update only certain properties. This is an asynchronous function.

```
Ctx.ERC721Token.updateToken(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to update.

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Token.updateToken(tokenAsset)
```

```
{
    "metadata": {
        "painting_name": "Mona_Lisa",
        "description": "Mona Lisa Painting",
        "image": "monalisa.jpeg",
        "painter_name": "Leonardo_da_Vinci"
    },
    "assetType": "otoken",
    "tokenId": "monalisa",
    "tokenName": "artcollection",
```

```
    "tokenDesc": "token description",
    "symbol": "ART",
    "tokenStandard": "erc721+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2022-04-05T08:30:42.000Z",
    "isBurned": false,
    "tokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
    "price": 100,
    "on_sale_flag": true
}
```

**getByRange**

This method calls the fabric `getStateByRange` method internally. Even though any asset with the given ID is returned from the ledger, this method casts the asset into the caller Asset type. This is an asynchronous static function.

```
@validator(yup.string(), yup.string())
public async getDigiCurrGetByRange(startId: string, endId: string) {
   return await this.Ctx.ERC721TOken.getByRange(startId, endId, PaintingNft);
}
```

```
Ctx.ERC721Token.getByRange(startId: string, endId: string,
tokenClassReference?: <Instance of Token Class> )
```

Parameters:

- `startId: string` – The starting key of the range. This key is included in the range.
- `endId: string` – The end key of the range. This key is excluded from the range.
- `tokenClassReference: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, a promise with an array of `<Token Class>`. On error, a rejection with an error message.

Return Value Example:


```
[
  {
      "metadata":{
         "painting_name":"Mona_Lisa",
         "description":"Mona Lisa Painting",
         "image":"monalisa.jpeg",
         "painter_name":"Leonardo_da_Vinci"
      },
      "assetType":"otoken",
      "tokenId":"monalisa",
      "tokenName":"artcollection",
      "tokenDesc":"token description",
      "symbol":"ART",
      "tokenStandard":"erc721+",
      "tokenType":"nonfungible",
      "tokenUnit":"whole",
      "behaviors":[
         "indivisible",
         "singleton",
         "mintable",
         "transferable",
         "burnable",
         "roles"
      ],
      "roles":{
         "minter_role_name":"minter"
      },
      "mintable":{
         "max_mint_quantity":20000
      },

"owner":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201
d347ad1288d",

"createdBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aa
e201d347ad1288d",

"transferredBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436d
d1aae201d347ad1288d",
      "creationDate":"2022-04-05T08:30:42.000Z",
      "transferredDate":"2022-04-05T09:28:30.000Z",
      "isBurned":false,
      "tokenUri":"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
      "price":100,
```

```
      "on_sale_flag":true
   }
]
```

**history**
This method returns history for the specified token. This is an asynchronous static function.

```
Ctx.ERC721Token.history(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a promise history query iterator for the specified token. On error, a rejection with an error message.

Return Value Example:

```
[
   {

"trxId":"e17a3154d5271be0492cbc7c12390b3480fec5a792d1cb1083e5335de56ebbd9",
      "timeStamp":1622614032,
      "isDelete":false,
      "value":{
         "metadata":{
            "painting_name":"Mona_Lisa",
            "description":"Mona Lisa Painting",
            "image":"monalisa.jpeg",
            "painter_name":"Leonardo_da_Vinci"
         },
         "assetType":"otoken",
         "tokenId":"monalisa",
         "tokenName":"artcollection",
         "tokenDesc":"token description",
         "symbol":"ART",
         "tokenStandard":"erc721+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "burnable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
         },
         "mintable":{
```

```
            "max_mint_quantity":20000
         },

"owner":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201
d347ad1288d",

"createdBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aa
e201d347ad1288d",
         "creationDate":"2022-04-05T08:30:42.000Z",
         "isBurned":false,
         "tokenUri":"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
         "price":100,
         "on_sale_flag":"true"
      }
   },
   {

"trxId":"dbcc4da410ad4d4a80996f090b313240f3f3d08aa2b5086afa8d0921f7b4c1
e5",
      "timeStamp":1622643853,
      "isDelete":false,
      "value":{
         "metadata":{
            "painting_name":"Mona_Lisa",
            "description":"Mona Lisa Painting",
            "image":"monalisa.jpeg",
            "painter_name":"Leonardo_da_Vinci"
         },
         "assetType":"otoken",
         "tokenId":"monalisa",
         "tokenName":"artcollection",
         "tokenDesc":"token description",
         "symbol":"ART",
         "tokenStandard":"erc721+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "burnable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
         },
         "mintable":{
            "max_mint_quantity":20000
         },
```

```
"owner":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad
1288d",

"createdBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d3
47ad1288d",

"transferredBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae2
01d347ad1288d",
         "creationDate":"2022-04-05T08:30:42.000Z",
         "transferredDate":"2022-04-05T09:28:30.000Z",
         "isBurned":false,
         "tokenUri":"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
         "price":100,
         "on_sale_flag":true
      }
   }
]
```

**getAllTokens**

This method returns all of the token assets that are saved in the state database. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This is an asynchronous static function.

```
Ctx.ERC721Token.getAllTokens()
```

Parameters:

- none

Returns:

- On success, a promise with all of the token assets. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Token.getAllTokens();

{
   "returnCode":"Success",
   "error":"",
   "result":{

"txid":"98e0a0a115803d25b843d630e6b23c435a192a03eb0a301fc9375f05da49a8b2",
      "payload":[
         "           "{
            "metadata":{
               "painting_name":"Mona_Lisa",
               "description":"Mona Lisa Painting",
               "image":"monalisa.jpeg",
               "painter_name":"Leonardo_da_Vinci"
            },
```

```
            "assetType":"otoken",
            "tokenId":"monalisa",
            "tokenName":"artcollection",
            "tokenDesc":"token description",
            "symbol":"ART",
            "tokenStandard":"erc721+",
            "tokenType":"nonfungible",
            "tokenUnit":"whole",
            "behaviors":[
               "indivisible",
               "singleton",
               "mintable",
               "transferable",
               "burnable",
               "roles"
            ],
            "roles":{
               "minter_role_name":"minter"
            },
            "mintable":{
               "max_mint_quantity":20000
            },

"owner":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201
d347ad1288d",

"createdBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aa
e201d347ad1288d",
            "creationDate":"2022-04-05T08:30:42.000Z",
            "isBurned":false,
            "tokenUri":"\"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg\"",
            "price":100,
            "on_sale_flag":false
         }"       "
      ],
      "encode":"JSON"
   }
}
```

**getAllTokensByUser**

This method returns all tokens that are owned by a specified account ID. This is an asynchronous static function.

```
Ctx.ERC721Token.getAllTokensByUser(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a promise history query iterator for the specified account. On error, a rejection with an error message.

Return Value Example:


```
{
   "returnCode":"Success",
   "error":"",
   "result":{

"txid":"98e0a0a115803d25b843d630e6b23c435a192a03eb0a301fc9375f05da49a8b2",
      "payload":[
         "           "{
            "metadata":{
               "painting_name":"Mona_Lisa",
               "description":"Mona Lisa Painting",
               "image":"monalisa.jpeg",
               "painter_name":"Leonardo_da_Vinci"
            },
            "assetType":"otoken",
            "tokenId":"monalisa",
            "tokenName":"artcollection",
            "tokenDesc":"token description",
            "symbol":"ART",
            "tokenStandard":"erc721+",
            "tokenType":"nonfungible",
            "tokenUnit":"whole",
            "behaviors":[
               "indivisible",
               "singleton",
               "mintable",
               "transferable",
               "burnable",
               "roles"
            ],
            "roles":{
               "minter_role_name":"minter"
            },
            "mintable":{
               "max_mint_quantity":20000
            },

"owner":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad
1288d",

"createdBy":"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d3
47ad1288d",
            "creationDate":"2022-04-05T08:30:42.000Z",
            "isBurned":false,
            "tokenUri":"\"https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg\"",
            "price":100,
```

```
            "on_sale_flag":false
         }"       "
      ],
      "encode":"JSON"
   }
```

**ownerOf**
This method returns the account ID of the owner of a specified token. This is an asynchronous static function.

```
Ctx.ERC721Token.ownerOf(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, it returns a promise history query iterator for the specified token ID. On error, it rejects with an error message

Return Value Example:

```
{"owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d"}
```

**tokenUri**
This method returns the URI for a specified token. This is an asynchronous static function.

```
Ctx.ERC721Token.tokenUri(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, it returns a promise history query iterator for the specified token ID. On error, it rejects with an error message

Return Value Example:

```
{"uri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"}
```

**getTokenUri**
This method returns the URI for a specified token. This is an asynchronous static function.

```
Ctx.ERC721Token.getTokenUri(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, it returns a promise history query iterator for the specified token ID. On error, it rejects with an error message

Return Value Example:

```
{"tokenUri": https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"}
```

**symbol**
This method returns the symbol of the token class.

```
Ctx.ERC721Token.symbol()
```

Parameters:

- none

Returns:

- On success, a JSON object with the token symbol.

Return Value Example:

```
{"symbol": "PNT"}
```

### Methods for Account Management

**generateAccountId**
This method returns an account ID, which is formed by concatenating the membership service provider ID (`orgId`) and the user name or email ID (`userId`) and then creating a SHA-256 hash.

```
Ctx.ERC721Account.generateAccountId(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a promise with the generated account ID. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Account.generateAccountId(orgId, userId)

oaccount~a0a60d54ba9e2ff349737d292ea10ebd9cc8f1991c11443c19d20aea299a9507
```

**`createAccount`**

This method creates an account for a specified user and token. An account must be created for any user who will have tokens at any point. Accounts track the number of NFTs a user has. Users must have accounts in the network to complete token-related operations. You can create only one NFT account per user.
An account ID is an alphanumeric set of characters, prefixed with `oaccount~` and followed by an SHA-256 hash of the membership service provider ID (`orgId`) of the user in the current network organization, the user name or email ID (`userId`) of the instance owner or the user who is logged in to the instance, and the constant string `nft`. This method can be called only by the `Token Admin` of the chaincode.

```
Ctx.ERC721Account.createAccount(orgId: string, userId: string,
tokenType: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenType: string` – The only supported token type is `nonfungible`.

Returns:

- On success, a promise with the new account object. On error, a rejection with an error message

Example:

```
await this.Ctx.ERC721Account.CreateAccount(orgId, userId, tokenType)

{
    "assetType": "oaccount",
    "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "userId": "admin",
    "orgId": "Org1MSP",
    "tokenType": "nonfungible",
    "noOfNfts": 0
}
```

**getAllAccounts**

This method returns a list of all accounts. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC721Account.getAllAccounts()
```

Parameters:

- none

Returns:

- On success, a promise with a JSON object that lists all accounts. On error, a rejection with an error message.

Return Value Example:

```
[
    {
        "key":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "valueJson": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    }
]
```

**history**

This method returns an array of the account history details for a specified account.

```
Ctx.ERC721Account.history(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a `map[string]interface{}` array that contains the account history details for the specified account. The account data is shown under the `value` key in the map. On error, a non-nil error object containing an error message.

Example:

```
await this.Ctx.ERC721Account.history(accountId)
```

```
[
    {
```

```
        "trxId":
"6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4",
        "timeStamp": 1649151044,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3ecf53a3",
        "timeStamp": 1649151022,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 2
        }
    },
    {
        "trxId":
"ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e7",
        "timeStamp": 1649150910,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9cb",
        "timeStamp": 1649149545,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
```

```
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 0
        }
    },
    {
        "trxId":
"e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c",
        "timeStamp": 1649147442,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 1
        }
    },
    {
        "trxId":
"d2d1f9c898707ae831e9361bc25da6369eac37b10c87dc04d18d6f3808222f08",
        "timeStamp": 1649137534,
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 0
        }
    }
]
```


**`getUserByAccountId`**

This method returns the user details for a specified account.

```
Ctx.ERC721Account.getUserByAccountId(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object that includes user details in the following properties:
  - `orgId` – The membership service provider (MSP) ID of the user in the current network organization.

  - `userId` – The user name or email ID of the user.
- On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Account.getUserByAccountById(accountId)
```

```
{
  "userId": "admin",
  "orgId": "Org1MSP"
}
```

**`getAccountWithStatusByUser`**

This method returns user details for a specified account, including account status. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC721Account.getAccountWithStatusByUser(orgId, userId)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user in the current organization.
- `userId` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `accountId` – The ID of the user account.
- `userId` – The user name or email ID of the user.
- `orgId` – The membership service provider (MSP) ID of the user in the current organization.
- `tokenType` – The type of token that the account holds.
- `noOfNfts` – The total number of NFTs held by the account.
- `bapAccountVersion` – An account object parameter for internal use.
- `status` – The current status of the user account.
- On error, a non-nil object that contains an error message.

Example:

```
await this.Ctx.ERC721Account.getAccountWithStatusByUser(orgId, userId)
```

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "accountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419
a9a",
```

```
  "userId": "idcqa",
  "orgId": "appdev",
  "tokenType": "nonfungible",
  "noOfNfts": 0
}
```

**getAccountByUser**

This method returns user details for a specified account. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC721Account.getAccountByUser(orgId, userId)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user in the current organization.
- `userId` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `accountId` – The ID of the user account.
- `userId` – The user name or email ID of the user.
- `orgId` – The membership service provider (MSP) ID of the user in the current organization.
- `tokenType` – The type of token that the account holds.
- `noOfNfts` – The total number of NFTs held by the account.
- On error, a non-nil object that contains an error message.

Example:

```
await this.Ctx.ERC721Account.getUserByAccountById(orgId, userId)
```

```
{
    "assetType": "oaccount",
    "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "userId": "admin",
    "orgId": "Org1MSP",
    "tokenType": "nonfungible",
    "noOfNfts": 0
}
```

**balanceOf**

This method returns the total number of NFTs the specified user holds.

```
Ctx.ERC721Account.balanceOf(accountId: string)
```

Parameters:

- `accountId: string` – The account ID of the user.

Returns:

- On success, a JSON object of the current NFT count. On error, a non-nil error object that contains an error message.

Example:

```
await this.Ctx.ERC721Account.balanceOf(accountId)
```

```
{"totalNfts": 0}
```

### Methods for Role Management

**`addRoleMember`**

This method adds a role to a specified user and token. An account ID is formed by creating an SHA-256 hash of the concatenated membership service provider ID (`orgId`) and the user name or email ID (`userId`). This is an asynchronous function.

```
Ctx.ERC721Token.addRoleMember(role: string, accountId: string)
```

Parameters:

- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `accountId: string` – The account ID to operate on.

Returns:

- On success, a promise with a message including the added role and account ID. On error, a rejection with an error message

Example:

```
await this.Ctx.ERC721Token.addRoleMember(role, accountId);
```

```
{"msg": "Successfully added role 'minter' to Account Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin)"}
```

**`removeRoleMember`**

This method removes a role from a specified user and token. An account ID is formed by creating an SHA-256 hash of the concatenated membership service provider ID (`orgId`) and the user name or email ID (`userId`). This is an asynchronous function.

```
Ctx.ERC721Token.removeRoleMember(role: string, accountId: string)
```

Parameters:

- `role: string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.

- `accountId: string` – The account ID to operate on.

Returns:

- On success, a promise with a message including the removed role and account ID. On error, a rejection with an error message

Example:

```
await this.Ctx.ERC721Token.removeRoleMember(role, accountId);
```

```
{"msg": "Successfully removed role 'minter' from Account Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1)"}
```

**`isInRole`**

This method returns a Boolean value to indicate if a user and token has a specified role. An account ID is formed by creating an SHA-256 hash of the concatenated membership service provider ID (`orgId`) and the user name or email ID (`userId`). This is an asynchronous function.

```
Ctx.ERC721Token.isInRole(role: string, accountId: string)
```

Parameters:

- `role: string` – The name of the role to check for the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `accountId: string` – The account ID to operate on.

Returns:

- On success, a promise that is true if the role is present for the specified account ID, otherwise false. On error, a rejection with an error message

Example:

```
await this.Ctx.ERC721Token.isInRole(role, accountId, tokenAsset)
```

```
{"result": "true"}
```

**`getAccountsByRole`**

This method returns a list of all account IDs for a specified role.

```
Ctx.ERC721Role.getAccountsByRole(roleName: string)
```

Parameters:

- `roleName: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs. On error, a non-nil error object that contains an error message.

Example:

```
await this.Ctx.ERC721Role.getAccountsByRole(userRole)
```

```
{
    "accounts": [

"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d"
    ]
}
```

**`getUsersByRole`**

This method returns a list of all users for a specified role.

```
Ctx.ERC721Role.getUsersByRole(userRole: string)
```

Parameters:

- `role: string` – The name of the role to search for.

Returns:

- On success, a JSON array of user objects. Each object contains the user ID and organization ID. On error, a non-nil error object that contains an error message.

Example:

```
await this.Ctx.ERC721Role.getUsersByRole(userRole)
```

```
{
    "users": [
        {
            "userId": "admin",
            "orgId": "Org1MSP"
        }
    ]
}
```

### Methods for Transaction History Management

**`getAccountTransactionHistory`**

This method returns an array of the transaction history details for a specified account.

```
Ctx.ERC721Account.getAccountTransactionHistory(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, an array of account transaction objects in JSON format:

- `transactionId` – The ID of the transaction.
- `transactedAccount` – The account with which the transaction took place.
- `transactionType` – The type of transaction.
- `timestamp` – The time of the transaction.
- `noOfNfts` – The balance of the caller account.
- On error, a non-nil error object that contains an error message.

Example:

```
await this.Ctx.ERC721Account.GetAccountTransactionHistory(accountId)

[
    {
        "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d
4",
        "timestamp": "2022-04-05T09:30:44.000Z",
        "tokenId": "monalisa1",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "BURN"
    },
    {
        "transactionId":
"otransaction~a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3ecf53a
3",
        "timestamp": "2022-04-05T09:30:22.000Z",
        "tokenId": "monalisa1",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e
7",
        "timestamp": "2022-04-05T09:28:30.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "CREDIT"
    },
    {
        "transactionId":
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9c
b",
        "timestamp": "2022-04-05T09:05:45.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "DEBIT"
```

```
    },
    {
        "transactionId":
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34
647337c",
        "timestamp": "2022-04-05T08:30:42.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "transactionType": "MINT"
    }
]
```

**`getAccountTransactionHistoryWithFilters`**
This method returns account transaction history for a specified user, filtered by `PageSize`, `Bookmark`, `startTime` and `endTime`. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
async getAccountTransactionHistoryWithFilters(orgId: string, userId:
string, filters?: Filters)
```

Parameters:

- `accountId: string` – The ID of the account.
- `filters: object` – An object of the Filter class that contains four attributes: `pageSize`, `bookmark`, `startTime` and `endTime`. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Returns:

- On success, an array of account transaction objects in JSON format:
  - `transactionId` – The ID of the transaction.
  - `transactedAccount` – The account with which the transaction took place.
  - `transactionType` – The type of transaction.
  - `timestamp` – The time of the transaction.
  - `noOfNfts` – The balance of the caller account.
  - On error, a non-nil error object that contains an error message.

Example:

```
await this.Ctx.ERC721Account.getAccountTransactionHistoryWithFilters(accountId,
filters)

[
    {
        "transactionId":
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e
7",
        "timestamp": "2022-04-05T09:28:30.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "CREDIT"
    },
    {
        "transactionId":
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9c
b",
        "timestamp": "2022-04-05T09:05:45.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
        "transactionType": "DEBIT"
    },
    {
        "transactionId":
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337
c",
        "timestamp": "2022-04-05T08:30:42.000Z",
        "tokenId": "monalisa",
        "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "transactionType": "MINT"
    }
]
```

**getTransactionById**

This method returns the history of a `Transaction` asset.

```
Ctx.ERC721Transaction.getTransactionById(transactionId: string)
```

Parameters:

- `transactionId: string` – The ID of the transaction asset.

Example:

```
await this.Ctx.ERC721Transaction.getTransactionById(transactionId)

{
    "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d
4",
```

```
    "history": [
        {
            "trxId":
"6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4",
            "timeStamp": 1649151044,
            "value": {
                "assetType": "otransaction",
                "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632
bdaf5d4",
                "tokenId": "monalisa1",
                "fromAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
                "toAccountId": "",
                "triggeredByAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
                "transactionType": "BURN",
                "timestamp": "2022-04-05T09:30:44.000Z",
            }
        }
    ]
}
```

**deleteHistoricalTransactions**

This method deletes transactions that are older than a specified date from the state database.

```
Ctx.ERC721Transaction.deleteTransactions(timeToExpiration: Date)
```

Parameters:

- `timeToExpiration: Date` – The date and time. Transactions older than the specified time will be deleted.

Example:

```
await this.Ctx.ERC721Transaction.deleteTransactions(timeToExpiration)
```

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"62ad6753cf2bfa54816b4c2f0ea325478b1cb1b84f8e13e6742c00f277310081",
        "payload": {
            "msg": "Successfuly deleted transaction older than date:
Fri Apr 08 2022 00:00:00 GMT+0000 (Coordinated Universal Time).",
            "transactions": [

"otransaction~e687531b71d943da2fb129638784fb93a96e7698013dfc51c8c6bf4f5
f797059",
```

```
"otransaction~18446adf59b669e12990a1cf3ea0a7a15764f967fa694cf263aee0cd5a21d952",

"otransaction~5560d4b5e0b0d0b9a6e97dcd7f81241a5daf56497a7b6819c6a55cebacc106f2",

"otransaction~f0a0a64ec1a0c92ac732706dd75ffbd3feecd9c48fc79e42c551485edf0542cb"
            ]
        },
        "encode": "JSON"
    }
}
```

### Token Behavior Management - Mintable Behavior

**`getMaxMintQuantity`**
This method returns the maximum mintable quantity of a token. If the `max_mint_quantity` behavior is not configured in the specification file, then the default value is 0 and an infinite number of tokens can be minted.

```
Ctx.ERC721Token.getMaxMintQuantity(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, the maximum mintable quantity of the token, in the number data type. On error, it returns with an error message.

Example:

```
await this.Ctx.ERC721Token.getMaxMintQuantity(tokenAsset);

20000
```

**`getTotalMintedTokens`**
This method returns the total minted number of tokens available in the system for the specified token. The net number of tokens available is the total number of minted tokens minus the number of burned tokens. This is an asynchronous function.

```
Ctx.ERC721Token.getTotalMintedTokens()
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to operate on.

Returns:

- On success, the total minted tokens, in the number data type. On error, it returns with an error message.

Example:
```
await this.Ctx.ERC721Token.getTotalMintedTokens(tokenAsset);

4000
```

### Token Behavior Management - Transferable Behavior

**`safeTransferFrom`**
This is an asynchronous function. This method transfers ownership of the specified NFT from the caller to another account. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.
- The caller of the function is the sender.

```
Ctx.ERC721Token.safeTransferFrom(fromAccountId: string, toAccountId:
string, token: <Instance of Token Class>, data?: string)
```

Parameters:

- `fromAccountId: string` – The account ID of the sender in the current organization.
- `toAccountId: string` – The account ID of the receiver in the current organization.
- `token: <Instance of Token Class>` – The token asset to transfer.
- `data: string` – Optional additional information to store in the transaction.

Returns:

- On success, a promise with a success message that includes account details. Account IDs have the prefix `oaccount~`. On error, a rejection with an error message.

Example:
```
await this.Ctx.ERC721Token.safeTransferFrom(fromAccountId, toAccountId,
tokenAsset, data);

{"msg": "Successfully transferred NFT token: 'monalisa' from Account-
Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729d
ba (Org-Id: Org1MSP, User-Id: user1)"}
```

**`transferFrom`**
This is an asynchronous function. This method transfers ownership of the specified NFT from a sender account to a receiver account. It is the responsibility of the caller

to pass the correct parameters. This method can be called by any user, not only the token owner. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.

```
Ctx.ERC721Token.transferFrom(fromAccountId: string, toAccountId: string,
token: <Instance of Token Class>)
```

Parameters:

- `fromAccountId: string` – The account ID of the sender in the current organization.
- `toAccountId: string` – The account ID of the receiver in the current organization.
- `token: <Instance of Token Class>` – The token asset to transfer.

Returns:

- On success, a promise with a success message that includes account details. Account IDs have the prefix `oaccount~`. On error, a rejection with an error message.

\Example:

```
await this.Ctx.ERC721Token.transferFrom(fromAccountId, toAccountId, tokeAsset,
data);
```

```
{"msg": "Successfully transferred NFT token: 'monalisa' from Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1) to Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin)"}
```

### Token Behavior Management - Burnable Behavior

**burn**

This method deactivates, or burns, the specified NFT from the caller's account. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a `burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) of the token can burn the token. This is an asynchronous function.

```
Ctx.ERC721Token.burn(token: <Instance of Token Class>)
```

Parameters:

- `token: <Instance of Token Class>` – The token asset to burn.

Returns:

- On success, a promise with a success message that includes account details. On error, a rejection with an error message.

Example:

```
await this.Ctx.ERC721Token.burn(tokenAsset);
```

```
{msg": "Successfully burned NFT token: 'monalisa1' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin)"}
```

## TypeScript Methods for ERC-721 NFT Locking

Blockchain App Builder automatically generates methods that you can use to lock non-fungible tokens that use the extended ERC-721 standard.

A locked token cannot be burned or transferred to other users. All other properties, such as the token's state, owner, and history are preserved. You can use the NFT locking functionality when transferring a token to another blockchain network, such as Ethereum or Polygon.

Before you can lock NFTs, you must assign the vault manager role to a user. The vault manager is a special type of role, a `TokenSys` role. `TokenSys` roles are different from asset-based roles such as burner, minter, and notary, and from administrative roles such as `Token Admin` and `Org Admin`. Currently Blockchain App Builder supports the `vault TokenSys` role. The single user who has the `vault` role for a chaincode is the vault manager of the chaincode, and can manage locked NFTs.

The typical flow for using the NFT locking functionality follows these steps.

- Create a non-fungible token that has the lockable behavior.
- Use the `addTokenSysRole` method to give the `vault` role to a user, the vault manager.
- Call the `lockNFT` method to lock a non-fungible token, specified by the token ID.

### TokenSys Role Management Methods

**`addTokenSysRole`**

This method adds a `TokenSys` role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async addTokenSysRole(role: string, orgId: string, userId:
string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.addTokenSysRoleMemb
er", "TOKEN");
    await
this.Ctx.ERC721Auth.checkAuthorization('ERC721TOKEN.addRoleMember',
'TOKEN');
    const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
    return await this.Ctx.ERC721Token.addTokenSysRoleMember(role,
```

```
accountId);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to give to the user.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully added role 'vault' to Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1)"
}
```

**`isInTokenSysRole`**

This method returns a Boolean value to indicate if a user has a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string())
public async isInTokenSysRole(orgId: string, userId: string, role: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.isInTokenSysRole",
"TOKEN", {orgId: orgId, userId: userId });
    const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
    return await this.Ctx.ERC721Token.isInTokenSysRole(role, accountId);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to check.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1) has vault role"
}
```

**removeTokenSysRole**
This method removes a `TokenSys` role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async removeTokenSysRole(role: string, orgId: string, userId:
string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.removeTokenSysRoleM
ember", "TOKEN");
    const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
    return await this.Ctx.ERC721Token.removeTokenSysRoleMember(role,
accountId);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to remove.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully removed role 'vault' from Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1)"
}
```

**transferTokenSysRole**
This method transfers a `TokenSys` role from a user to another user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string())
```

```
public async transferTokenSysRole(role: string, fromOrgId: string,
fromUserId: string, toOrgId: string, toUserId: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.transferTokenSysRole",
"TOKEN");
    const fromAccountId = await
this.Ctx.ERC721Account.generateAccountId(fromOrgId, fromUserId);
    const toAccountId = await
this.Ctx.ERC721Account.generateAccountId(toOrgId, toUserId);
    return await this.Ctx.ERC721Token.transferTokenSysRole(role,
fromAccountId, toAccountId);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to transfer.
- `fromOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role from.
- `fromUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role from.
- `toOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role to.
- `toUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role to.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully transfered role 'vault' from Account Id:
ouaccount~f4e311528f03fffa7810753d643f66289ff6c9080fcf839902f28a1d3aff1789
(Org-Id: Org1MSP, User-Id: user1) to Account Id:
ouaccount~ae5be2ae8f98d6d32f5d02b43877d987114e7937c7bacbc30390dcce09996a19
(Org-Id: Org1MSP, User-Id: user2)"
}
```

**`getAccountsByTokenSysRole`**

This method returns a list of all account IDs for a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getAccountsByTokenSysRole(role: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.getAccountsByTokenSysRole
", "TOKEN");
```

```
    return await this.Ctx.ERC721Token.getAccountsByTokenSysRole(role);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "accountIds": [

"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03
eba"
    ]
}
```

**getUsersByTokenSysRole**

This method returns user information for all users with a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getUsersByTokenSysRole(role: string) {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.getUsersByTokenSysR
ole", "TOKEN");
    return await this.Ctx.ERC721Token.getUsersByTokenSysRole(role);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
 "users":[
      {

"accountId":"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb8
3c874c2caf03eba",
         "orgId":"Org1MSP",
         "userId":"user1"
      }
```

```
    ]
}
```

### NFT Locking Methods

**`lockNFT`**
This method locks a specified non-fungible token. To lock a token, there must be a user with the `TokenSys vault` role, who acts as the vault manager. This method can be called only by the owner of the token.

```
@Validator(yup.string())
public async lockNFT(tokenId: string) {
    return await this.Ctx.ERC721Token.lockNFT(tokenId);
}
```

Parameters:

- `tokenID: string` – The ID of the token to lock.

Returns:

- On success, a JSON representation of the token object.

Return Value Example:

```
{
   "assetType":"otoken",
   "tokenId":"token1",
   "tokenName":"artcollection",
   "symbol":"ART",
   "tokenStandard":"erc721+",
   "tokenType":"nonfungible",
   "tokenUnit":"whole",
   "behaviors":[
      "indivisible",
      "singleton",
      "mintable",
      "transferable",
      "lockable",
      "burnable",
      "roles"
   ],
   "roles":{
      "minter_role_name":"minter"
   },
   "mintable":{
      "max_mint_quantity":20000
   },

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
   "creationDate":"2023-10-20T10:26:29.000Z",
```

```
"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff
1b6a7733463",
   "isBurned":false,
   "isLocked":true,
   "tokenUri":"token1.example.com",
   "price":120,
   "on_sale_flag":false
}
```

**isNFTLocked**

This method returns a Boolean value to indicate if a specified token is locked. This method can be called only by the token owner, the vault manager (the user with the `TokenSys vault` role), or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async isNFTLocked(tokenId: string) {
  try {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.isNFTLocked",
"TOKEN", { tokenId });
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC721Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VAU
LT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
  const isLocked = await this.Ctx.ERC721Token.isNFTLocked(tokenId);
  return {isLocked: isLocked}
}
```

Parameters:

- `tokenID: string` – The ID of the token.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "isNFTLocked":true
}
```

**getAllLockedNFTs**

This method returns a list of all locked non-fungible tokens. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async getAllLockedNFTs() {
  try {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.getAllLockedNFTs",
"TOKEN");
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC721Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VAULT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
 return this.Ctx.ERC721Token.getAllLockedNFTs();
}
```

Parameters:

- None

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:

```
[
   {
      "key":"token1",
      "valueJson":{
         "assetType":"otoken",
         "tokenId":"token1",
         "tokenName":"artcollection",
         "symbol":"ART",
         "tokenStandard":"erc721+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "lockable",
            "burnable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
```

```
         },
         "mintable":{
            "max_mint_quantity":20000
         },

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304
ebff1b6a7733463",
         "creationDate":"2023-10-20T10:26:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff
1b6a7733463",
         "isBurned":false,
         "isLocked":true,
         "tokenUri":"token1.example.com",
         "price":120,
         "on_sale_flag":false
      }
   }
]
```

**getAllLockedNFTsByOrg**

This method returns a list of all locked non-fungible tokens for a specified organization and optionally a specified user. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getLockedNFTsByOrg(orgId: string, userId?: string) {
  try {
    await
this.Ctx.ERC721Auth.checkAuthorization("ERC721TOKEN.getLockedNFTsByOrg"
, "TOKEN");
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC721Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VAU
LT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
  return await this.Ctx.ERC721Token.getLockedNFTsByOrg(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user (optional).

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:


```
[
   {
      "key":"token1",
      "valueJson":{
         "assetType":"otoken",
         "tokenId":"token1",
         "tokenName":"artcollection",
         "symbol":"ART",
         "tokenStandard":"erc721+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "lockable",
            "burnable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
         },
         "mintable":{
            "max_mint_quantity":20000
         },

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "creationDate":"2023-10-20T10:26:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
         "isBurned":false,
         "isLocked":true,
         "tokenUri":"token1.examplecom",
         "price":120,
         "on_sale_flag":false
      }
   }
]
```


## TypeScript Methods for ERC-721 Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for tokens that use the extended ERC-721 standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, and transferring tokens. Additionally, other users cannot

transfer tokens to a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.

- Automatically Generated Account Status Methods
- Account Status SDK Methods

### Automatically Generated Account Status Methods

**`getAccountStatus`**
This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
@Validator(yup.string(), yup.string())
     public async getAccountStatus(orgId: string, userId: string) {
       const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
       await
this.Ctx.ERC721Auth.checkAuthorization("ERC721ACCOUNT_STATUS.get",
"TOKEN", { accountId });
       try {
         return await
this.Ctx.ERC721AccountStatus.getAccountStatus(accountId);
       } catch (err) {
        return await
this.Ctx.ERC721AccountStatus.getDefaultAccountStatus(accountId);
       }
     }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "accountId":
```

```
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

**getAccountStatusHistory**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
public async getAccountStatusHistory(orgId: string, userId: string) {
       const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
       await this.Ctx.ERC721Account.getAccount(accountId);
       await
this.Ctx.ERC721Auth.checkAuthorization("ERC721ACCOUNT_STATUS.history",
"TOKEN", { accountId });
       const status_id = await
this.Ctx.ERC721AccountStatus.generateAccountStatusId(accountId);
       let accountStatusHistory: any;
       try {
         accountStatusHistory = await
this.Ctx.ERC721AccountStatus.history(status_id);
       } catch (err) {
         return [];
       }
       return accountStatusHistory;
     }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, the account status history in JSON format.

Return Value Example:

```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "suspended"
```

```
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "active"
    }
  }
]
```

**activateAccount**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode. Deleted accounts cannot be activated. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
@Validator(yup.string(), yup.string())
 public async activateAccount(orgId: string, userId: string) {
   await
this.Ctx.ERC721Auth.checkAuthorization("ERC721ACCOUNT_STATUS.activateAc
count", "TOKEN");
   const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
   return await this.Ctx.ERC721Account.activateAccount(accountId);
 }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
```

```
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "active"
}
```

**suspendAccount**

This method suspends a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
@Validator(yup.string(), yup.string())
 public async suspendAccount(orgId: string, userId: string) {
   await
this.Ctx.ERC721Auth.checkAuthorization("ERC721ACCOUNT_STATUS.suspendAccount",
 "TOKEN");
   const accountId = await this.Ctx.ERC721Account.generateAccountId(orgId,
userId);
   return await this.Ctx.ERC721Account.suspendAccount(accountId);
 }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "suspended"
}
```

**deleteAccount**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot be changed to any other state. To delete an account, the account balance must be zero.

```
@Validator(yup.string(), yup.string())
 public async deleteAccount(orgId: string, userId: string) {
   await
```

```
this.Ctx.ERC721Auth.checkAuthorization("ERC721ACCOUNT_STATUS.deleteAcco
unt", "TOKEN");
   const accountId = await
this.Ctx.ERC721Account.generateAccountId(orgId, userId);
   return await this.Ctx.ERC721Account.deleteAccount(accountId);
 }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
  "status": "deleted"
}
```

### Account Status SDK Methods

**`getAccountStatus`**
This method gets the current status of the token account.

```
Ctx.ERC721AccountStatus.getAccountStatus(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
```

```
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

**`getAccountStatusHistory`**
This method gets the history of the account status.

```
Ctx.ERC721AccountStatus.history(statusId: string)
```

Parameters:

- `statusId: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:

```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "suspended"
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "active"
    }
  }
]
```

**activateAccount**

This method activates a token account. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
Ctx.ERC721Account.activateAccount(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
  "status": "active"
}
```

**suspendAccount**

This method suspends a token account.

```
Ctx.ERC721AccountStatus.suspendAccount(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
```

```
    "status": "suspended"
}
```

**deleteAccount**
This method deletes a token account.

```
Ctx.ERC721Account.deleteAccount(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

## Scaffolded Go NFT Project for ERC-721

Blockchain App Builder takes the input from your NFT specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates NFT lifecycle classes and functions, including CRUD and non-CRUD methods. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to NFTs, see Scaffolded Go Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated NFT Methods
  - Custom Methods
- NFT SDK Methods

### Model

Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class.

```
package model

type ArtCollection struct {
    AssetType     string `json:"AssetType" final:"otoken"`
    TokenId       string `json:"TokenId" id:"true" mandatory:"true"
validate:"regexp=^[A-Za-z0-9][A-Za-z0-9_-]*$,max=16"`
    TokenName     string `json:"TokenName" final:"artcollection"`
    TokenDesc     string `json:"TokenDesc" validate:"max=256"`
    Symbol        string `json:"Symbol" final:"ART"`
    TokenStandard string `json:"TokenStandard" final:"erc721+"`
    TokenType     string `json:"TokenType" final:"nonfungible"
validate:"regexp=^nonfungible$"`
    TokenUnit     string `json:"TokenUnit" final:"whole"
validate:"regexp=^whole$"`

    Behavior []string `json:"Behavior"
final:"[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"b
urnable\",\"roles\"]"`

    Roles map[string]interface{} `json:"Roles"
final:"{\"minter_role_name\":\"minter\"}"`

    Mintable map[string]interface{} `json:"Mintable"
final:"{\"Max_mint_quantity\":20000}"`

    Owner           string `json:"Owner,omitempty" validate:"string"`
    CreatedBy       string `json:"CreatedBy,omitempty"
validate:"string"`
    TransferredBy   string `json:"TransferredBy,omitempty"
validate:"string"`
    CreationDate    string `json:"CreationDate,omitempty"
validate:"string"`
    TransferredDate string `json:"TransferredDate,omitempty"
validate:"string"`
    IsBurned        bool   `json:"IsBurned" validate:"bool"`
    BurnedBy        string `json:"BurnedBy,omitempty"
validate:"string"`
    BurnedDate      string `json:"BurnedDate,omitempty"
validate:"string"`
    TokenUri        string `json:"TokenUri" validate:"string,max=2000"`

    TokenMetadata ArtCollectionMetadata `json:"TokenMetadata"`

    Price        int  `json:"Price" validate:"int"`
    On_sale_flag bool `json:"On_sale_flag" validate:"bool"`
}

type ArtCollectionMetadata struct {
    Painting_name string `json:"Painting_name" validate:"string"`
```

```
    Description   string `json:"Description" validate:"string"`
    Image         string `json:"Image" validate:"string"`
    Painter_name  string `json:"Painter_name" validate:"string"`
}
```

### Controller

There is only one main controller.

```
type Controller struct {
    Ctx trxcontext.TrxContext
}
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

### Automatically Generated NFT Methods

Blockchain App Builder automatically generates methods to support NFTs and NFT life cycles. You can use these methods to initialize NFTs, manage roles and accounts, and complete other NFT lifecycle tasks without any additional coding.

Blockchain App Builder automatically generates methods to support NFTs and NFT life cycles. You can use these methods to initialize NFTs, manage roles and accounts, and complete other NFT lifecycle tasks without any additional coding. Controller methods must be public to be invokable. Public method names begin with an upper case character. Method names that begin with a lower case character are private.

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

**`AddTokenAdmin`**
This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenAdmin(orgId string, userId string)
(interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ADMIN.AddAdmin", "TOKEN")
```

```
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Admin.AddAdmin(orgId, userId)
}
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully added Admin (OrgId: Org1MSP, UserId: user1)"}
```

**RemoveTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. You cannot use this method to remove yourself as a `Token Admin`.

```
func (t *Controller) RemoveTokenAdmin(orgId string, userId string)
(interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ADMIN.RemoveAdmin", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Admin.RemoveAdmin(orgId, userId)
}
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully removed Admin (OrgId Org1MSP UserId user1)"}
```

**IsTokenAdmin**

This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`. A `Token Admin` can call this function on any other user in the blockchain network. Other users can call this method only on their own accounts.

```
func (t *Controller) IsTokenAdmin(orgId string, userId string) (interface{},
error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ADMIN.IsTokenAdmin", "TOKEN",
map[string]string{"orgId": orgId, "userId": userId})
      if err != nil || !auth {
            return false, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Auth.IsUserTokenAdmin(orgId, userId)
}
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

Return Value Example:

```
{"result":true}
```

**GetAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) GetAllTokenAdmins() (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ADMIN.GetAllAdmins", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Admin.GetAllAdminUsers()
}
```

Parameters:

- none

Returns:

- On success, a JSON list of admins that includes `OrgId` and `UserId` objects.

Return Value Example:

```
{
    "admins": [
        {
            "OrgId":"Org1MSP",
            "UserId":"admin"
        }
    ]
}
```

### Methods for Token Configuration Management

**`Init`**

This method is called when the chaincode is instantiated. Every `Token Admin` is identified by the `UserId` and `OrgId` information in the `adminList` parameter. The `UserId` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `OrgId` is the membership service provider (MSP) ID of the user in the current network organization. The `adminList` parameter is mandatory the first time you deploy the chaincode. If you are upgrading the chaincode, pass an empty list (`[]`). Any other information in the `adminList` parameter is ignored during upgrades.

```
func (t *Controller) Init(adminList
[]erc721Admin.ERC721TokenAdminAsset) (interface{}, error) {
      list, err := t.Ctx.ERC721Admin.InitAdmin(adminList)
      if err != nil {
            return nil, fmt.Errorf("initialising admin list failed
%s", err.Error())
      }
      <Token Name> := <Token Class>{}
      _, err = t.Ctx.ERC721Token.SaveClassInfo(&<Token Name>)
      if err != nil {
            return nil, err
      }
      _, err = t.Ctx.ERC721Token.SaveDeleteTransactionInfo()
      if err != nil {
            fmt.Println("error: ", err)
      }
      return list, nil
}
```

Parameters:

- `adminList array` – An array of `{OrgId, UserId}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

**GetAllTokens**

This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAllTokens() (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetAllTokens", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Token.GetAllTokens()
}
```

Parameters:

- none

Returns:

- A list of all token assets in JSON format.

Return Value Example:

```
[
    {
        "key": "monalisa",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "CreationDate": "2022-04-10T11:01:42Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 0,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
```

```
            "Symbol": "PNT",
            "TokenDesc": "token Description",
            "TokenId": "monalisa",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
        }
    },
    {
        "key": "monalisa2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "CreationDate": "2022-04-10T11:04:44Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "monalisa1",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
```

```
                "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
        }
        }
    }
]
```

**GetAllTokensByUser**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetAllTokensByUser(orgId string, userId string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetAllTokensByUser",
"TOKEN", map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
    return t.Ctx.ERC721Token.GetAllTokensByUser(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- A list of token assets in JSON format.

Return Value Example:

```
[
    {
        "key": "monalisa",
        "valueJson": {
```

```
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "CreationDate": "2022-04-10T11:01:42Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 0,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "token Description",
            "TokenId": "monalisa",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
        }
    },
    {
        "key": "monalisa2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
```

```
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "CreationDate": "2022-04-10T11:04:44Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "monalisa1",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
        }
        }
    }
]
```

**GetTokenById**

This method returns a token object if the token is present in the state database. This method can be called only by a `Token Admin` of the chaincode or the token owner.

```
func (t *Controller) GetTokenById(tokenId string) (interface{}, error) {
      auth, err := t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.Get",
"TOKEN", map[string]string{"tokenId": tokenId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      tokenAsset, err := t.getTokenObject(tokenId)
      if err != nil {
            return nil, err
      }
```

```
      return tokenAsset.Interface(), nil
}
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Returns:

- The token asset in JSON format.

Return Value Example:


```
{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-04-06T00:35:42Z",
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "NftBasePrice": 200,
    "Owner":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "Token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
```

```
    "TransferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "TransferredDate": "2022-04-06T00:51:56Z"
}
```

**GetTokenHistory**

This method returns the history for a specified token ID. This method can only be called when connected to the remote Oracle Blockchain Platform network. Anyone can call this method.

```
func (t *Controller) GetTokenHistory(tokenId string) (interface{}, error) {
      /*
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetTokenHistory", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      */
      return t.Ctx.ERC721Token.History(tokenId)
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
[
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T19:22:52z",
        "TxId":
"6b7989be322956164a8d1cd7bf2a7187d59eba73ce756e6bf946ab48b349bbc0",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "CreationDate": "2022-04-06T19:22:23z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
```

```
            "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "token Description",
            "TokenId": "monalisa",
            "TokenMetadata": {
               "Description": "Mona Lisa Painting",
               "Image": "monalisa.jpeg",
               "PainterName": "Leonardo_da_Vinci",
               "PaintingName": "Mona_Lisa"
            },
           "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg""
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T19:22:23z",
        "TxId":
"e61bcb3cb61c8920f7e6d8f0d19726c7c88d876e0ad6cfb052cfb92d49985c3f",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "CreationDate": "2022-04-06T19:22:23z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
            "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "Roles": {
```

```
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            ""TokenId": "monalisa",
            "TokenMetadata": {
                "Description": "Mona Lisa Painting",
                "Image": "monalisa.jpeg",
                "PainterName": "Leonardo_da_Vinci",
                "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"
        }
    }
]
```

**getTokenObject**

This is a utility method that returns an instance of the token for a specified token ID. This method is used by many of the automatically generated methods to fetch token objects. You can call this method as needed from your custom methods. When you create a tokenized asset or class, update the switch case with the corresponding `Token` class to return the correct token object. The `ochain sync` command in Blockchain App Builder automatically creates a switch case when a tokenized asset is created in the specification file. Because this method is private, it is not directly invokable and can only be called from other methods.

```
func (t *Controller) getTokenObject(tokenId string) (reflect.Value, error) {
      if tokenId == "" {
            return reflect.Value{}, fmt.Errorf("error in retrieving token,
token_id cannot be empty")
      }
      tokenAsset, err := t.Ctx.ERC721Token.Get(tokenId)
      if err != nil {
            return reflect.Value{}, fmt.Errorf("no token exists with id %s
%s", tokenId, err.Error())
      }
      tokenName := tokenAsset.(map[string]interface{})["TokenName"].(string)
      switch tokenName {
      case "<Token Name>":
            var asset <Token Class>
            _, err := t.Ctx.ERC721Token.Get(tokenId, &asset)
            if err != nil {
                  return reflect.Value{}, err
            }
            return reflect.ValueOf(&asset), nil
      default:
```

```
            return reflect.Value{}, fmt.Errorf("no token exists with
token name %s", tokenName)
      }
}
```

Parameters:

- `tokenId: string` – The ID of the token.

**`OwnerOf`**

This method returns the account ID of the owner of the specified token ID. Anyone can call this method.

```
func (t *Controller) Ownerof(tokenId string) (interface{}, error) {
      return t.Ctx.ERC721Token.OwnerOf(tokenId)
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- A JSON object of the owner's account ID.

Return Value Example:

```
{"Owner":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba"}
```

**`Name`**

This method returns the name of the token class. Anyone can call this method.

```
func (t *Controller) Name() (interface{}, error) {
      return t.Ctx.ERC721Token.Name()
}
```

Parameters:

- none

Returns:

- A JSON object of the token name.

Return Value Example:

```
{"TokenName": "paintingnft"}
```

**`Symbol`**
This method returns the symbol of the token class. Anyone can call this method.

```
func (t *Controller) Symbol() (interface{}, error) {
      return t.Ctx.ERC721Token.Symbol()
}
```

Parameters:

- none

Returns:

- A JSON object of the token symbol.

Return Value Example:

```
{"Symbol": "PNT"}
```

**`TokenURI`**
This method returns the URI of a specified token. Anyone can call this method.

```
func (t *Controller) TokenURI(tokenId string) (interface{}, error) {
      return t.Ctx.ERC721Token.TokenURI(tokenId)
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object of the token URI.

Return Value Example:

```
{"TokenURI": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"}
```

**`TotalSupply`**
This method returns the total number of minted tokens. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TotalSupply() (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.TotalSupply", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
```

```
      return t.Ctx.ERC721Token.TotalSupply()
}
```

Parameters:

- none

Returns:

- On success, a JSON object of the token count.

Return Value Example:

```
{"TotalSupply": 3}
```

**`TotalNetSupply`**
This method returns the total number of minted tokens minus the number of burned tokens. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TotalNetSupply() (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.TotalNetSupply",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Token.GetTotalMintedTokens()
}
```

Parameters:

- none

Returns:

- On success, a JSON object of the token count.

Return Value Example:

```
{"TotalNetSupply": 2}
```

### Methods for Account Management

**`CreateAccount`**
This method creates an account for a specified user and token. An account must be created for any user who will have tokens at any point. Accounts track the number of NFTs a user has. Users must have accounts in the network to complete token-related operations. You can create only one NFT account per user.
An account ID is an alphanumeric set of characters, prefixed with `oaccount~` and followed by an SHA-256 hash of the membership service provider ID (`OrgId`) of the user in the current network organization, the user name or email ID (`UserId`) of the

instance owner or the user who is logged in to the instance, and the constant string `nft`. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) CreateAccount(orgId string, userId string, tokenType
string) (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.CreateAccount", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Account.CreateAccount(orgId, userId, tokenType)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenType: string` – The only supported token type is `nonfungible`.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
    "AssetType": "oaccount",
    "BapAccountVersion" : 0,
    "AccountId":
"oaccount~a0a60d54ba9e2ff349737d292ea10ebd9cc8f1991c11443c19d20aea299a9507",
    "UserId": "admin",
    "OrgId": "Org1MSP",
    "TokenType": "nonfungible",
    "NoOfNfts": 0
}
```

**`BalanceOf`**

This method returns the total number of NFTs that a specified user holds. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) BalanceOf(orgId string, userId string) (interface{},
error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.BalanceOf", "TOKEN",
map[string]string{"orgId": orgId, "userId": userId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
```

```
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
      if err != nil{
            return nil,err
      }
      return t.Ctx.ERC721Account.BalanceOf(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- A JSON object of the current NFT count.

Return Value Example:

```
{"totalNfts": 0}
```

**GetAllAccounts**

This method returns a list of all accounts. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAllAccounts() (interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.GetAllAccounts",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Account.GetAllAccounts()
}
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
    {
        "key":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
```

```
fd1",
        "valueJson": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 5,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    },
    {
        "key":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
        "valueJson": {
            "AccountId":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user_minter"
        }
    },
    {
        "key":
"oaccount~5541fb520058d83664b844e7a55fe98d574ddeda765d0e795d4779e9ccc271ce",
        "valueJson": {
            "AccountId":
"oaccount~5541fb520058d83664b844e7a55fe98d574ddeda765d0e795d4779e9ccc271ce",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user_burner"
        }
    }
]
```

**GetAccountByUser**

This method returns account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
func (t *Controller) GetAccountByUser(orgId string, userId string)
(interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.GetAccountByUser",
"TOKEN", map[string]string{"orgId": orgId, "userId": userId})
      if err != nil && !auth {
```

```
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Account.GetAccountWithStatusByUser(orgId,
userId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `AccountId` – The ID of the user account.
- `UserId` – The user name or email ID of the user.
- `OrgId` – The membership service provider (MSP) ID of the user in the current organization.
- `TokenType` – The type of token that the account holds.
- `NoOfNfts` – The total number of NFTs held by the account.
- `BapAccountVersion` – An account object parameter for internal use.
- `Status` – The current status of the user account.

Return Value Example:

```
{
  "AccountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419
a9a",
  "AssetType": "oaccount",
  "BapAccountVersion": 0,
  "NoOfNfts": 0,
  "OrgId": "appdev",
  "Status": "active",
  "TokenType": "nonfungible",
  "UserId": "idcqa"
}
```

**GetUserByAccountId**

This method returns the user details of a specified account. This method can be called by any user.

```
func (t *Controller) GetUserByAccountId(accountId string)
(interface{}, error) {
      return t.Ctx.ERC721Account.GetUserByAccountById(accountId)
}
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`OrgId` and `UserId`).

Return Value Example:

```
{
    "OrgId": "Org1MSP",
    "UserId": "admin"
}
```

**GetAccountHistory**

This method returns account history for a specified user. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetAccountHistory(orgId string, userId string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.History", "TOKEN",
map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Account.History(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a list of objects.

Return Value Example:

```
[
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T08:16:53Z",
        "TxId":
"750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf7",
```

```
        "Value": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "AssetType": "oaccount",
            "BapAccountVersion" : 1,
            "NoOfNfts": 1,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T08:15:19Z",
        "TxId":
"49eb84c42d452e5ba0028d8ebb4190454cf9013a11c0bad3e96594af452d4982",
        "Value": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    }
]
```

### Methods for Role Management

**`AddRole`**

This method adds a role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddRole(role string, orgId string, userId string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.AddRoleMember",
"TOKEN", map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
```

```
      return t.Ctx.ERC721Token.AddRoleMember(role, accountId)
}
```

Parameters:

- `userRole: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message with user details.

Return Value Example:

```
{"msg": "Successfully added role minter to
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(orgId : Org1MSP, userId : admin)"}
```

**RemoveRole**

This method removes a role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) RemoveRole(userRole string, orgId string, userId
string) (interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.RemoveRoleMember", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Token.RemoveRoleMember(userRole, accountId)
}
```

Parameters:

- `userRole: string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message with user details.

Return Value Example:

```
{"msg": "successfully removed memberId
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (orgId : Org1MSP, userId : admin) from role minter"}
```

**GetAccountsByRole**

This method returns a list of all account IDs for a specified role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetAccountsByRole(userRole string) (interface{},
error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ROLE.GetAccountsByRole",
"TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      return t.Ctx.ERC721Role.GetAccountsByRole(userRole)
}
```

Parameters:

- `userRole: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs.

Return Value Example:

```
{
    "accounts": [

"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d"
    ]
}
```

**GetUsersByRole**

This method returns a list of all users for a specified role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetUsersByRole(userRole string) (interface{},
error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ROLE.GetUsersByRole",
"TOKEN")
      if err != nil && !auth {
```

```
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      return t.Ctx.ERC721Role.GetUsersByRole(userRole)
}
```

Parameters:

- `userRole: string` – The name of the role to search for.

Returns:

- On success, a JSON array of the user objects (`orgId` and `userId`).

Return Value Example:

```
{
    "Users": [
        {
            "OrgId": "Org1MSP",
            "UserId": "admin"
        }
    ]
}
```

**IsInRole**

This method returns a Boolean value to indicate if a user has a specified role. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
func (t *Controller) IsInRole(orgId string, userId string, role string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.IsInRole", "TOKEN",
map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      result, err := t.Ctx.ERC721Token.IsInRole(role, accountId)
      if err != nil {
            return nil, fmt.Errorf("error in IsInRole %s", err.Error())
      }
      response := make(map[string]interface{})
      response["result"] = result
      return response, nil
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `userRole: string` – The name of the role to search for.

Returns:

- On success, a JSON string of the Boolean result.

Return Value Example:

```
{"result":"true"}
```

### Methods for Transaction History Management

**`GetAccountTransactionHistory`**
This method returns account transaction history for a specified user. This method can be called by the `Token Admin` of the chaincode or the owner of the account.

```
func (t *Controller) GetAccountTransactionHistory(orgId string, userId
string) (interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.GetAccountTransactio
nHistory", "TOKEN", map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }

      transactionArray, err :=
t.Ctx.ERC721Account.GetAccountTransactionHistory(accountId)
      return transactionArray, err
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a list of objects.

Return Value Example:

```
[
     {
        "Timestamp": "2022-04-06T08:31:39Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
        "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
7",
        "TransactionType": "DEBIT"
    }
    {
        "Timestamp": "2022-04-06T08:16:53Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
        "TransactionId":
"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf
7",
        "TransactionType": "MINT"
    }
]
```

**GetAccountTransactionHistoryWithFilters**

This method returns account transaction history for a specified user, filtered by `PageSize`, `Bookmark`, `StartTime` and `EndTime`. This method can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetAccountTransactionHistoryWithFilters(orgId string,
userId string, filters ...erc721Account.AccountHistoryFilters) (interface{},
error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, err
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.GetAccountTransactionHisto
ryWithFilters", "TOKEN", map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }

      transactionArray, err :=
t.Ctx.ERC721Account.GetAccountTransactionHistoryWithFilters(accountId,
filters...)
      return transactionArray, err
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Returns:

- On success, a list of objects.

Return Value Example:

```
[
     {
        "Timestamp": "2022-04-06T08:31:39Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502c
ec4",
        "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a4
40e8ac7",
        "TransactionType": "DEBIT"
    }
    {
        "Timestamp": "2022-04-06T08:16:53Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
        "TransactionId":
"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76
392ecf7",
        "TransactionType": "MINT"
    }
]
```

**GetTransactionById**

This method returns transaction history for a specified transaction ID. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetTransactionById(transactionId string)
(interface{}, error) {
      return t.Ctx.ERC721Transaction.GetTransactionById(transactionId)
}
```

Parameters:

- `transactionId: string` – The id of the transaction asset.

Returns:

- On success, a list of objects.

Return Value Example:

```
{
    "History": [
        {
            "IsDelete": "false",
            "Timestamp": "2022-04-06T08:31:39Z",
            "TxId":
"5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac7",
            "Value": {
                "AssetType": "otransaction",
                "Data": "",
                "FromAccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
                "Timestamp": "2022-04-06T08:31:39Z",
                "ToAccountId":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
                "TokenId": "monalisa",
                "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
7",
                "TransactionType": "TRANSFER",
                "TriggeredByAccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1"
            }
        }
    ],
    "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
7"
}
```

**`DeleteHistoricalTransactions`**

This method deletes transactions older than a specified time stamp in the state database. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) DeleteHistoricalTransactions(timestamp string)
(interface{}, error) {
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TRANSACTION.DeleteHistoricalTransa
ctions", "TOKEN")
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
```

```
      return
t.Ctx.ERC721Transaction.DeleteHistoricalTransactions(timestamp)
}
```

Parameters:

- `timestamp: Date` – A time stamp. All transactions before the time stamp will be deleted.

Returns:

- On success, a list of objects.

Return Value Example:

```
{
    "Transactions": [

"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76
392ecf7"
    ],
    "msg": "Successfuly deleted transaction older than
date:2022-04-06T08:17:53Z"
}
```

### Methods for Token Behavior Management - Mintable Behavior

**`Create<Token Name>Token`**
This method creates (mints) an NFT. The asset and associated properties are saved in the state database. The caller of this transaction must have a token account. The caller of this transaction becomes the owner of the NFT. If the token specification file includes the `roles` section for `behaviors` and the `minter_role_name` property for `roles`, then the caller of the transaction must have the minter role. Otherwise, any caller can mint NFTs.

```
func (t *Controller) Create<Token Name>Token(tokenAsset <Token Class>)
(interface{}, error) {
      return t.Ctx.ERC721Token.CreateToken(&tokenAsset)
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to mint. For more information about the properties of the token asset, see the input specification file.

Returns:

- On success, a JSON token asset object that includes the following properties:
- `Behavior` – A description of all token behaviors.
- `CreatedBy` – The account ID of the user who called the transaction to mint the token.
- `CreationDate` – The time stamp of the transaction.

- `IsBurned` – A Boolean value that indicates if the NFT identified by `tokenId` is burned.
- `Mintable` – A description of the properties of mintable behavior. The `max_mint_quantity` property specifies the maximum number of NFTs of this token class that can be created.
- `Owner` – The account ID of the current owner of the token. During the minting process, the caller of this method becomes the owner of the token.
- `Symbol` – The symbol of the token.
- `TokenDesc` – The description of the token.
- `TokenMetadata` – JSON information that describes the token.
- `TokenName` – The name of the token.
- `TokenStandard` – The standard of the token.
- `TokenType` – The type of token held by this account.
- `TokenUnit` – The unit of the token.
- `TokenUri` – The URI of the token.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",

    "CreationDate": "2022-04-06T08:16:53Z",
    "IsBurned": false,
     "Mintable": {
        "Max_mint_quantity": 20000
    },
    "NftBasePrice": 100,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
```

```
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"
}
```

**Update<Token Name>Token**

This method updates token properties. This method can be called only by the user who is the owner or creator of the token. After a token asset is created, only the token owner can update the token custom properties. If the user is both token owner and creator of a token, they can also update the `TokenDesc` property. Token metadata cannot be updated. You must pass all token properties to this method, even if you want to update only certain properties.

```
func (t *Controller) Update<Token Name>Token(tokenAsset <Token Class>)
(interface{}, error) {
      return t.Ctx.ERC721Token.UpdateToken(&tokenAsset)
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to update. For more information about the the properties of the token asset, see the input specification file.

Returns:

- On success, an updated JSON token asset object

Return Value Example:

```
{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-04-06T08:16:53Z",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
```

```
    },
    "NftBasePrice": 200,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "Token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`SafeTransferFrom`**

This method transfers ownership of the specified NFT from the caller to another account. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.
- The caller of the function is the sender.

```
func (t *Controller) SafeTransferFrom(fromOrgId string, fromUserId string,
toOrgId string, toUserId string, tokenId string, data ...string)
(interface{}, error) {
      tokenAssetValue, err := t.getTokenObject(tokenId)
      if err != nil {
            return nil, err
      }
      fromAccountId, err := t.Ctx.ERC721Account.GenerateAccountId(fromOrgId,
fromUserId)
      if err != nil {
            return nil, err
      }
      toAccountId, err := t.Ctx.ERC721Account.GenerateAccountId(toOrgId,
toUserId)
```

```
      if err != nil {
            return nil, err
      }
      return t.Ctx.ERC721Token.SafeTransferFrom(fromAccountId,
toAccountId, tokenAssetValue.Interface(), data...)
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender in the current organization.
- `fromUserId: string` – The user name or email ID of the sender.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenId: string` – The ID of the token to transfer.
- `data: string` – Optional additional information to store in the transaction record.

Returns:

- On success, a message with the sender and receiver account details.

Return Value Example:

```
{"msg": "Successfully transferred NFT token: 'monalisa' from Account-
Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729d
ba (Org-Id: Org1MSP, User-Id: user1)"}
```

**`TransferFrom`**

This method transfers ownership of the specified NFT from a sender account to a receiver account. It is the responsibility of the caller to pass the correct parameters. This method can be called by any user, not only the token owner. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.

```
func (t *Controller) TransferFrom(fromOrgId string, fromUserId string,
toOrgId string, toUserId string, tokenId string) (interface{}, error) {
      tokenAssetValue, err := t.getTokenObject(tokenId)
      if err != nil {
            return nil, err
      }
      fromAccountId, err :=
t.Ctx.ERC721Account.GenerateAccountId(fromOrgId, fromUserId)
```

```
	if err != nil {
		return nil, err
	}
	toAccountId, err := t.Ctx.ERC721Account.GenerateAccountId(toOrgId,
toUserId)
	if err != nil {
		return nil, err
	}
	return t.Ctx.ERC721Token.TransferFrom(fromAccountId, toAccountId,
tokenAssetValue.Interface())
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender in the current organization.
- `fromUserId: string` – The user name or email ID of the sender.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenId: string` – The ID of the token to transfer.

Returns:

- On success, a message with the sender and receiver account details.

Return Value Example:

```
{"msg": "Successfully transferred NFT token: 'monalisa' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1)"}
```

### Methods for Token Behavior Management - Burnable Behavior

**`Burn`**

This method deactivates, or burns, the specified NFT from the caller's account. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a `burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) of the token can burn the token.

```
func (t *Controller) Burn(tokenId string) (interface{}, error) {
	tokenAssetValue, err := t.getTokenObject(tokenId)
	if err != nil {
		return nil, err
	}
```

```
      return t.Ctx.ERC721Token.Burn(tokenAssetValue.Interface())
}
```

Parameters:

- `tokenId: string` – The ID of the token to burn.

Returns:

- On success, a message with the account details.

Return Value Example:

```
{"msg": "Successfully burned NFT token: 'monalisa' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin"}
```

**BurnNFT**

This method deactivates, or burns, the specified NFT from the caller's account, and returns a token object and token history. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a `burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) or owner of the token can burn the token.

```
func (t *Controller) BurnNFT(tokenId string) (interface{}, error) {
      tokenAsset, err := t.Ctx.ERC721Token.Get(tokenId)
      if err != nil {
            return nil, err
      }
      tokenHistory, err := t.Ctx.ERC721Token.History(tokenId)
      if err != nil {
            return nil, err
      }
      token := tokenAsset.(map[string]interface{})
      if token[constants.IsBurned] == true {
            return nil, fmt.Errorf("token with tokenId %s is already
burned", tokenId)
      }
      token[constants.TokenId], err =
strconv.Atoi(token[constants.TokenId].(string))
      if err != nil {
            return nil, fmt.Errorf("tokenId is expected to be integer
but found %s", tokenId)
      }
      tokenHistoryBytes, err := json.Marshal(tokenHistory)
      if err != nil {
            return nil, err
      }
      var tokenHistoryAsRawJson json.RawMessage
      err = json.Unmarshal(tokenHistoryBytes, &tokenHistoryAsRawJson)
```

```
        if err != nil {
                return nil, err
        }
        token[constants.TokenHistory] = string(tokenHistoryAsRawJson)
        token[constants.IsBurned] = true
        _, err = t.Burn(tokenId)
        if err != nil {
                return nil, err
        }
        return token, nil
}
```

Parameters:

- `tokenId: string` – The ID of the token to burn.

Returns:

- On success, a token object that includes token history information.

Return Value Example:

```
{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2023-08-22T13:19:33+05:30",
    "IsBurned": true,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Price": 120,
    "Roles": {
        "minter_role_name": "minter"
    },
    "Symbol": "ART",
    "TokenDesc": "",
    "TokenHistory":
"[{\"IsDelete\":\"false\",\"Timestamp\":\"2023-08-22T13:19:33+05:30\",\"TxId\
":\"0219099bcaaecd5f76f7f08d719384fd5ed34103a55bd8d3754eca0bfc691594\",\"Valu
e\":{\"AssetType\":\"otoken\",\"Behavior\":
[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"burnable\",\"r
oles\"],\"CreatedBy\":\"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436
```

```
dd1aae201d347ad1288d\",\"CreationDate\":\"2023-08-22T13:19:33+05:30\",\
"IsBurned\":false,\"Mintable\":
{\"Max_mint_quantity\":20000},\"On_sale_flag\":false,\"Owner\":\"oaccou
nt~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d\",\
"Price\":120,\"Roles\":
{\"minter_role_name\":\"minter\"},\"Symbol\":\"ART\",\"TokenDesc\":\"\"
,\"TokenId\":\"1\",\"TokenMetadata\":
{\"Description\":\"\",\"Image\":\"\",\"Painter_name\":\"\",\"Painting_n
ame\":\"\"},\"TokenName\":\"artcollection\",\"TokenStandard\":\"erc721+
\",\"TokenType\":\"nonfungible\",\"TokenUnit\":\"whole\",\"TokenUri\":\
"example.com\"}}]",
    "TokenId": 1,
    "TokenMetadata": {
        "Description": "",
        "Image": "",
        "Painter_name": "",
        "Painting_name": ""
    },
    "TokenName": "artcollection",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "example.com"
}
```

### Custom Methods

You can use the token SDK methods to write custom methods for your business application.

The following example shows how to use token SDK methods in custom methods. When the `Sell` method is called, it posts a token for sale for a specified price.

```
func (t *Controller) Sell(tokenId string, sellingPrice int)
(interface{}, error) {
    var tokenAsset ArtCollection
    _, err := t.Ctx.ERC721Token.Get(tokenId, &tokenAsset)
    if err != nil {
       return nil, err
    }

 /**  * price is a custom asset
      attribute to set the price of a non-fungible token in the
      marketplace  */
    tokenAsset.Price = sellingPrice
 /**  * on_sale_flag is a
      custom asset attribute that maintains non-fungible token selling
status in the
      marketplace  */
    tokenAsset.On_sale_flag = true

    _, err = t.Ctx.ERC721Token.UpdateToken(tokenAsset)
    if err != nil {
        return nil, err
    }
```

```
    msg := fmt.Sprintf("Token ID : %s has been posted for selling in the
marketplace", tokenId)
    return msg, nil
}
```

### NFT SDK Methods

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

The NFT SDK provides an access control function. Some methods can be called only by a `Token Admin` or `Account Owner` of the token. You can use this feature to ensure that operations are carried out only by the intended users. Any unauthorized access results in an error. To use the access control function, import the `Authorization` class from the `../lib/auth` module.

```
import { ERC721Authorization } from '../lib/erc721-auth';
```

**`CheckAuthorization`**
Use this method to add an access control check to an operation. Most automatically generated methods include access control. Certain token methods can be run only by an `ERC721Admin` or the `Account Owner` of the token or by the `MultipleAccountOwner` for users with multiple accounts. The `CheckAuthorization` method is part of the `erc721Auth` package, which you access via the `Ctx` struct (receiver). The access control mapping is described in the `oChainUtil.go` file, as shown in the following text. You can modify access control by editing the `oChainUtil.go` file. To use your own access control or to disable access control, remove the access control code from the automatically generated controller methods and custom methods.

```
var t TokenAccess
    var r RoleAccess
    var a AccountAccess
    var as AccountStatusAccess
    var h HoldAccess
    var ad AdminAccess
    var trx TransactionAccess
    var tc TokenConversionAccess
    var auth AuthAccess
```

```
      var erc721ad ERC721AdminAccess
      var erc721t ERC721TokenAccess
      var erc721r ERC721RoleAccess
      var erc721a ERC721AccountAccess
      var erc721as ERC721AccountStatusAccess
      var erc721trx ERC721TransactionAccess
     auth.IsTokenAdmin = []string{"Admin", "MultipleAccountOwner"}

      trx.DeleteHistoricalTransactions = []string{"Admin"}
      ad.AddAdmin = []string{"Admin"}
      ad.RemoveAdmin = []string{"Admin"}
      ad.GetAllAdmins = []string{"Admin", "OrgAdmin"}
      ad.AddOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
      ad.RemoveOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
      ad.GetOrgAdmins = []string{"Admin", "OrgAdmin"}
      ad.IsTokenAdmin = []string{"Admin", "MultipleAccountOwner",
"OrgAdmin"}
      t.Save = []string{"Admin"}
      t.GetAllTokens = []string{"Admin", "OrgAdmin"}
      t.Update = []string{"Admin"}
      t.GetTokenDecimals = []string{"Admin", "OrgAdmin"}
      t.GetTokensByName = []string{"Admin", "OrgAdmin"}
      t.AddRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
      t.RemoveRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
      t.IsInRole = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      t.GetTotalMintedTokens = []string{"Admin", "OrgAdmin"}
      t.GetNetTokens = []string{"Admin", "OrgAdmin"}
      t.Get = []string{"Admin", "OrgAdmin"}
      t.GetTokenHistory = []string{"Admin", "OrgAdmin"}
      a.CreateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      a.AssociateToken = []string{"Admin", "OrgAdminAccountIdCheck"}
      a.GetAllAccounts = []string{"Admin"}
      a.GetAllOrgAccounts = []string{"Admin", "OrgAdminOrgIdCheck"}
      a.GetAccount = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      a.History = []string{"Admin", "AccountOwner"}
      a.GetAccountTransactionHistory = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetAccountTransactionHistoryWithFilters = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetSubTransactionsById = []string{"Admin",
"TransactionInvoker"}
      a.GetSubTransactionsByIdWithFilters = []string{"Admin",
"TransactionInvoker"}
      a.GetAccountBalance = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetAccountOnHoldBalance = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
      a.GetOnHoldIds = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      a.GetAccountsByUser = []string{"Admin", "OrgAdminOrgIdCheck",
"MultipleAccountOwner"}
```

```
        as.Get = []string{"Admin", "OrgAdminAccountIdCheck", "AccountOwner"}
        as.ActivateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
        as.SuspendAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
        as.DeleteAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
        r.GetAccountsByRole = []string{"Admin"}
        r.GetUsersByRole = []string{"Admin"}
        r.GetOrgAccountsByRole = []string{"Admin", "OrgAdminOrgIdCheck"}
        r.GetOrgUsersByRole = []string{"Admin", "OrgAdminOrgIdCheck"}

        tc.InitializeExchangePoolUser = []string{"Admin"}
        tc.AddConversionRate = []string{"Admin"}
        tc.UpdateConversionRate = []string{"Admin"}
        tc.GetConversionRate = []string{"Admin", "OrgAdmin", "AnyAccountOwner"}
        tc.GetConversionRateHistory = []string{"Admin", "OrgAdmin",
"AnyAccountOwner"}
        tc.TokenConversion = []string{"Admin", "AnyAccountOwner"}
        tc.GetExchangePoolUser = []string{"Admin"}
        erc721ad.AddAdmin = []string{"Admin"}
        erc721ad.GetAllAdmins = []string{"Admin"}
        erc721ad.IsTokenAdmin = []string{"Admin"}
        erc721ad.RemoveAdmin = []string{"Admin"}
        erc721trx.DeleteHistoricalTransactions = []string{"Admin"}
        erc721t.Save = []string{"Admin"}
        erc721t.GetAllTokens = []string{"Admin"}
        erc721t.Update = []string{"Admin"}
        erc721t.GetTokensByName = []string{"Admin"}
        erc721t.AddRoleMember = []string{"Admin"}
        erc721t.RemoveRoleMember = []string{"Admin"}
        erc721t.IsInRole = []string{"Admin", "AccountOwner"}
        erc721t.Get = []string{"Admin", "TokenOwner"}
        erc721t.GetAllTokensByUser = []string{"Admin", "AccountOwner"}
        erc721t.TotalSupply = []string{"Admin"}
        erc721t.TotalNetSupply = []string{"Admin"}
        erc721t.History = []string{"Admin"}

        erc721a.CreateAccount = []string{"Admin"}
        erc721a.CreateUserAccount = []string{"Admin"}
        erc721a.CreateTokenAccount = []string{"Admin"}
        erc721a.AssociateFungibleTokenToAccount = []string{"Admin",
"AccountOwner"}
        erc721a.GetAllAccounts = []string{"Admin"}
        erc721a.History = []string{"Admin", "AccountOwner"}
        erc721a.GetAccountTransactionHistory = []string{"Admin",
"AccountOwner"}
        erc721a.GetAccountTransactionHistoryWithFilters = []string{"Admin",
"AccountOwner"}
        erc721a.GetAccountByUser = []string{"Admin", "MultipleAccountOwner"}
        erc721a.BalanceOf = []string{"Admin", "MultipleAccountOwner"}
        erc721as.Get = []string{"Admin", "AccountOwner"}
        erc721as.ActivateAccount = []string{"Admin"}
        erc721as.SuspendAccount = []string{"Admin"}
        erc721as.DeleteAccount = []string{"Admin"}
```

```
erc721r.GetAccountsByRole = []string{"Admin"}
erc721r.GetUsersByRole = []string{"Admin"}

var accessMap TokenAccessControl
accessMap.Token = t
accessMap.Account = a
accessMap.AccountStatus = as
accessMap.Hold = h
accessMap.Role = r
accessMap.Admin = ad
accessMap.Auth = auth
accessMap.TokenConversion = tc
accessMap.ERC721ADMIN = erc721ad
accessMap.ERC721TOKEN = erc721t
accessMap.ERC721ACCOUNT = erc721a
accessMap.ERC721AccountStatus = erc721as
accessMap.ERC721ROLE = erc721r
accessMap.ERC721TRANSACTION = erc721trx
```

```
Ctx.ERC721Auth.CheckAuthorization(funcName string, args []string)
(bool, error)
```

Parameters:

- `funcName: string` – The map value between the receivers and methods as described in the `oChainUtil.go` file.
- `...args` – A variable argument where `args[0]` takes the constant `'TOKEN'` and `args[1]` takes the `accountId` parameter to add an access control check for an `AccountOwner`. To add an access control check for a `MultipleAccountOwner`, `args[1]` takes the `orgId` parameter and `args[2]` takes the `userId` parameter. To add an access control check for a `TokenOwner`, `args[1]` takes the `tokenId` parameter.

Returns:

- A Boolean response and, if required, an error.

Examples:

```
t.Ctx.ERC721Auth.CheckAuthorization(<parameters>)
```

`Admin` access

```
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetAllTokens",
"TOKEN")
```

`AccountOwner` access

```
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.History", "TOKEN",
accountId)
```

`MultipleAccountOwner` access

```
t.Ctx.ERC721Auth.CheckAuthorization("ERC721ACCOUNT.BalanceOf", "TOKEN",
orgId, userId)
```

`TokenOwner` access

```
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.Get", "TOKEN", tokenId)
```

**`IsUserTokenAdmin`**
This method returns a map with the Boolean value `true` if the caller of the function is a `Token Admin`. Otherwise the method returns `false`.

```
Ctx.ERC721Auth.IsUserTokenAdmin(orgId string, userId string)  (interface{},
error)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user in the current network organization.
- `userId` – The user name or email ID of the user.

Returns:

- A Boolean response.

Example:

```
t.Ctx.ERC721Auth.IsUserTokenAdmin(orgId, userId)

{"result":true}
```

**`AddAdmin`**
This method adds a user as a `Token Admin` of the token chaincode.

```
Ctx.ERC721Admin.AddAdmin(orgId string, userId string) (interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user added as a `Token Admin` of the token chaincode. On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Admin.AddAdmin(orgId, userId)

{"msg":"Successfully added Admin (OrgId: Org1MSP, UserId: user1)"}
```

**`RemoveAdmin`**
This method removes a user as a `Token Admin` of the token chaincode.

```
Ctx.ERC721Admin.RemoveAdmin(orgId string, userId string) (interface{},
error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user removed as a `Token Admin` of the token chaincode. On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Admin.RemoveAdmin(orgId, userId)

{"msg":"Successfuly removed Admin (OrgId Org1MSP UserId user1)"}
```

**`GetAllAdmins`**
This method returns a list of all `Token Admin` users.

```
Ctx.ERC721Admin.GetAllAdmins() (Admin[], error)
```

Parameters:

- none

Returns:

- On success, a list of all `Token Admin` users. On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Admin.GetAllAdmins()

{
    "admins": [
```

```
        {
            "OrgId":"Org1MSP",
            "UserId":"admin"
        }
    ]
}
```

**`GetAllAdminUsers`**
This method returns a list of all `Token Admin` users.

```
Ctx.ERC721Admin.GetAllAdminUsers() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a list of all `Token Admin` users in `map[string]interface{}` format. On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Admin.GetAllAdminUsers()
```

```
{
    "admins": [
        {
            "OrgId":"Org1MSP",
            "UserId":"admin"
        }
    ]
}
```

### Methods for Token Configuration Management

**`CreateToken`**
This method creates a token and saves its properties in the state database. This method can be called only by users with the minter role.

```
Ctx.ERC721Token.CreateToken(args ...interface{})
```

Parameters:

- A variable argument where `args[0]` contains a reference to the token `struct` of the required type.

Returns:

- On success, an `interface[]` with token details. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.CreateToken(&tokenAsset)

{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-04-06T08:16:53Z",
    "IsBurned": false,
     "Mintable": {
        "Max_mint_quantity": 20000
    },
    "NftBasePrice": 100,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"
}
```

**GetTokenUri**

This method returns the token URI for a specified token.

```
Ctx.ERC721Token.GetTokenURI(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, it returns a map of the new token URI in the string data type. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.GetTokenURI(tokenId)

{"TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"}
```

**`TokenUri`**

This method returns the token URI for a specified token.

```
Ctx.ERC721Token.TokenURI(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, it returns a map of the new token URI in the string data type. On error, a non-nil object with an error message.

Return Value Example:

```
t.Ctx.ERC721Token.TokenURI(tokenId)

{"TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"}
```

**`Symbol`**

This method returns the symbol of the token class.

```
Ctx.ERC721Token.Symbol() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a map of the symbol in the string data type. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.Symbol()

{"Symbol": "PNT"}
```

**Name**
This method returns the name of the token class.

```
Ctx.ERC721Token.Name() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a map of the token name in the string data type. On error, a non-nil object containing an error message.

Example:
```
t.Ctx.ERC721Token.Name()

{"TokenName": "paintingnft"}
```

**OwnerOf**
This method returns the account ID of the owner of a specified token.

```
Ctx.ERC721Token.OwnerOf(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a map of the account ID of the owner in the string data type. On error, a non-nil object containing an error message.

Example:
```
t.Ctx.ERC721Token.OwnerOf(tokenId)

{"Owner":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba"}
```

**TotalSupply**
This method returns the total number of minted NFTs.

```
Ctx.ERC721Token.TotalSupply() (map[string]interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a map of the total supply of tokens in the number data type. On error, a rejection with an error message.

Example:

```
t.Ctx.ERC721Token.TotalSupply();

{"TotalSupply": 3}
```

**`GetAllTokens`**

This method returns all of the token assets that are saved in the state database. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC721Token.GetAllTokens()  (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a map of all of the token assets. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.GetAllTokens()

[
    {
        "key": "monalisa",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintab

le",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "CreationDate": "2022-04-10T11:01:42Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
```

```
            "NftBasePrice": 0,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "token Description",
            "TokenId": "monalisa",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
        }
    },
    {
        "key": "monalisa2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "CreationDate": "2022-04-10T11:04:44Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
```

```
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "monalisa1",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
        }
        }
    }
]
```

**GetAllTokensByUser**

This method returns all tokens that are owned by a specified account ID. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC721Token.GetAllTokensByUser(accountId string) (interface{}, error)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a map of token assets for the specified account. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.GetAllTokensByUser(accountId)
```

```
[
    {
        "key": "monalisa",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
```

```
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "CreationDate": "2022-04-10T11:01:42Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 0,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "token Description",
            "TokenId": "monalisa",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
        }
    },
    {
        "key": "monalisa2",
        "valueJson": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "CreationDate": "2022-04-10T11:04:44Z",
            "IsBurned": false,
            "Mintable": {
```

```
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 100,
            "Owner":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "monalisa1",
            "TokenMetadata": {
                 "Description": "Mona Lisa Painting",
                 "Image": "monalisa.jpeg",
                 "PainterName": "Leonardo_da_Vinci",
                 "PaintingName": "Mona_Lisa"
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg",
        }
        }
    }
]
```

**Get**

This method returns the specified token object if it is present in the state database.

```
Ctx.Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.
- `result` – A variable argument, where the first argument `result[0]` is a reference to an empty `Token` object of the correct type, which will contain the token data after a successful call of the method.

Returns:

- On success, a map with the token asset data. Also, if `result[0]` is passed then the data is assigned to `result[0]`. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.Get(tokenId, &asset)
```

```
{
    "AssetType": "otoken",
```

```
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-04-06T00:35:42z",
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "NftBasePrice": 200,
    "Owner":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "Token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
    "TransferredBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "TransferredDate": "2022-04-06T00:51:56z"
}
```


**UpdateToken**

This method updates token properties. This method can be called only by the owner or creator of the token. After a token asset is created, only the token owner can update the token custom properties. If the user is both token owner and creator of a token, they can also update the `TokenDesc` property. Token metadata cannot be

updated. You must pass all token properties to this method, even if you want to update only certain properties.

```
Ctx.ERC721Token.UpdateToken(asset interface{}) (interface{}, error)
```

Parameters:

- A reference to the token struct data of the required type

Returns:

- On success, a promise message with token details. On error, a rejection with an error message.

Example:

```
t.Ctx.ERC721Token.UpdateToken(&asset)

{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",

    "CreationDate": "2022-04-06T08:16:53Z",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "NftBasePrice": 200,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "Symbol": "PNT",
    "TokenDesc": "Token Description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "paintingnft",
    "TokenStandard": "erc721+",
```

```
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg",
}
```

**History**

This method returns history for the specified token.

```
Ctx.ERC721Token.History(tokenId: string)  (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, an array of maps. On error, a rejection with an error message.

Example:

```
t.Ctx.ERC721Token.History(tokenId)

[
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T11:34:06z",
        "TxId":
"3184eac8738c73ef45501fe23c9e14517892e04e4eb03ec9be834b89c29ea17b",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "BurnedBy": null,
            "BurnedDate": null,
            "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "CreationDate": "2022-04-06T11:33:40+05:30",
            "IsBurned": null,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 0,
            "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
```

```
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "t1",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "PainterName": "",
                "PaintingName": ""
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "",
            "TransferredBy": null,
            "TransferredDate": null
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T11:33:40z",
        "TxId":
"d37dba907a849c308b2a38d47cf8a68cdcb4e3d93fa74050774379fccfcd43be",
        "Value": {
            "AssetType": "otoken",
            "Behavior": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "CreationDate": "2022-04-06T11:33:40+05:30",
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "NftBasePrice": 0,
            "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "Symbol": "PNT",
            "TokenDesc": "",
            "TokenId": "t1",
```

```
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "PainterName": "",
                "PaintingName": ""
            },
            "TokenName": "paintingnft",
            "TokenStandard": "erc721+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": ""
        }
    }
]
```

**GetNewCtx**

This method returns a new `TrxContext` object. The `trxcontext` struct holds references to all of the SDK libraries. Access the sdk methods by using only this object. The `trxcontext` object maintains the mutual exclusivity of transaction stubs in SDK libraries when concurrent transactions are running.

```
GetNewCtx(stub shim.ChaincodeStubInterface) TrxContext
```

Parameters:

- `stub` – The transaction stub.

Returns:

- A `trxcontext` struct.

Example:

```
trxcontext.GetNewCtx(stub)

trxcontext object.
```

### Methods for Account Management

**GenerateAccountId**

This method returns an account ID, which is formed by concatenating the membership service provider ID (`orgId`) and the user name or email ID (`userId`) and then creating a SHA-256 hash.

```
Ctx.ERC721Account.GenerateAccountId(orgId string, userId string)
(string, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, the generated account ID. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)

oaccount~a0a60d54ba9e2ff349737d292ea10ebd9cc8f1991c11443c19d20aea299a9507
```

**`CreateAccount`**

This method creates an account for a specified user. An account must be created for any user who will have tokens at any point. Accounts track the number of NFTs a user has. Users must have accounts in the network to complete token-related operations. You can create only one NFT account per user.

An account ID is an alphanumeric set of characters, prefixed with `oaccount~` and followed by an SHA-256 hash of the membership service provider ID (`org_id`) of the user in the current network organization, the user name or email ID (`userId`) of the instance owner or the user who is logged in to the instance, and the constant string `nft`.

```
Ctx.ERC721Account.CreateAccount(orgId string, userId string, tokenType
string) (ERC721Account, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenType: string` – The only supported token type is `nonfungible`.

Returns:

- On success, the new account object. On error, a non-nil object containing an error message

Example:

```
t.Ctx.ERC721Account.CreateAccount(orgId, userId, tokenType)

{
    "AssetType": "oaccount",
    "AccountId":
"oaccount~a0a60d54ba9e2ff349737d292ea10ebd9cc8f1991c11443c19d20aea299a9507",
    "UserId": "admin",
    "BapAccountVersion" : 0,
    "OrgId": "Org1MSP",
    "TokenType": "nonfungible",
    "NoOfNfts": 0
}
```

**GetAllAccounts**

This method returns a list of all accounts. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC721Account.GetAllAccounts() (interface{}, error)
```

Parameters:

- none

Returns:

- A JSON array of all accounts.

Example:

```
t.Ctx.ERC721Account.GetAllAccounts()

[
    {
        "key":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
        "valueJson": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "BapAccountVersion" : 0,
            "AssetType": "oaccount",
            "NoOfNfts": 5,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    },
    {
        "key":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502c
ec4",
        "valueJson": {
            "AccountId":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502c
ec4",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user_minter"
        }
    },
    {
        "key":
```

```
"oaccount~5541fb520058d83664b844e7a55fe98d574ddeda765d0e795d4779e9ccc271ce",
        "valueJson": {
            "AccountId":
"oaccount~5541fb520058d83664b844e7a55fe98d574ddeda765d0e795d4779e9ccc271ce",
            "AssetType": "oaccount",
            "BapAccountVersion" : 0,
            "NoOfNfts": 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user_burner"
        }
    }
]
```

**History**

This method returns an array of the account history details for a specified account.

```
Ctx.ERC721Account.History(accountId string) (interface{}, error)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a `map[string]interface{}` array that contains the account history details for the specified account. The account data is shown under the `value` key in the map. On error, a non-nil error object containing an error message.

Example:

```
t.Ctx.ERC721Account.History(accountId)
```

```
[
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T08:16:53Z",
        "TxId":
"750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf7",
        "Value": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
            "AssetType": "oaccount",
            "BapAccountVersion" : 1,
            "NoOfNfts": 1,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2022-04-06T08:15:19Z",
        "TxId":
```

```
"49eb84c42d452e5ba0028d8ebb4190454cf9013a11c0bad3e96594af452d4982",
        "Value": {
            "AccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
            "AssetType": "oaccount",
            "NoOfNfts": 0,
            "BapAccountVersion" : 0,
            "OrgId": "apPart",
            "TokenType": "nonfungible",
            "UserId": "user1"
        }
    }
]
```

**GetUserByAccountId**
This method returns the user details for a specified account.

```
Ctx.ERC721Account.GetUserByAccountId(accountId string) (interface{},
error)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object that includes user details in the following properties:
  - `OrgId` – The membership service provider (MSP) ID of the user in the current network organization.
  - `UserId` – The user name or email ID of the user.
- On error, a non-nil object containing an error message.

Example:
`t.Ctx.ERC721Account.GetUserByAccountById(accountId)`

```
{
    "OrgId": "Org1MSP",
    "UserId": "admin"
}
```

**GetAccountWithStatusByUser**
This method returns account details for a specified user, including account status. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC721Account.GetAccountWithStatusByUser(orgId, userId)
(interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `AccountId` – The ID of the user account.
- `UserId` – The user name or email ID of the user.
- `OrgId` – The membership service provider (MSP) ID of the user in the current organization.
- `TokenType` – The type of token that the account holds.
- `NoOfNfts` – The total number of NFTs held by the account.
- `BapAccountVersion` – An account object parameter for internal use.
- `Status` – The current status of the user account.
- On error, a non-nil object that contains an error message.

Example:

```
{
  "AccountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
  "AssetType": "oaccount",
  "BapAccountVersion": 0,
  "NoOfNfts": 0,
  "OrgId": "appdev",
  "Status": "active",
  "TokenType": "nonfungible",
  "UserId": "idcqa"
}
```

**GetAccountByUser**

This method returns account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC721Account.GetAccountByUser(orgId, userId) (ERC721Account, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes the following properties:
- `AccountId` – The ID of the user account.

- `UserId` – The user name or email ID of the user.
- `OrgId` – The membership service provider (MSP) ID of the user in the current organization.
- `TokenType` – The type of token that the account holds.
- `NoOfNfts` – The total number of NFTs held by the account.
- On error, a non-nil object that contains an error message.

Example:

```
{
    "AssetType": "oaccount",
    "AccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "BapAccountVersion" : 0,
    "UserId": "admin",
    "OrgId": "Org1MSP",
    "TokenType": "nonfungible",
    "NoOfNfts": 0
}
```

**`BalanceOf`**

This method returns the total number of NFTs the specified user holds.

```
Ctx.ERC721Account.BalanceOf(accountId string) (interface{}, error)
```

Parameters:

- `accountId: string` – The account ID of the user.

Returns:

- On success, an interface with a message and the total number of NFTs. On error, a non-nil error object that contains an error message.

Example:

```
t.Ctx.ERC721Account.BalanceOf(accountId)

{"TotalNfts": 0}
```

### Methods for Role Management

**`AddRoleMember`**

This method adds a role to a specified user.

```
Ctx.ERC721Token.AddRoleMember(role string, accountId string)
(interface{}, error)
```

Parameters:

- `role: string` – The name of the role to add to the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `accountId: number` – The account ID to operate on.

Returns:

- On success, a map with a success message. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.AddRoleMember(userRole, accountId)
```

```
{"msg": "Successfully added role minter to
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(orgId : Org1MSP, userId : admin)"}
```

**`RemoveRoleMember`**

This method removes a role from a specified user and token. An account ID is formed by creating an SHA-256 hash of the concatenated membership service provider ID (`orgId`) and the user name or email ID (`userId`).

```
Ctx.Token.RemoveRoleMember(role string, accountId string) (interface{},
error)
```

Parameters:

- `role: string` – The name of the role to remove from the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `accountId: number` – The account ID to operate on.

Returns:

- On success, a map with a success message. On error, a non-nil object containing an error message.

Example:

```
t.Ctx.ERC721Token.RemoveRoleMember(userRole, accountId)
```

```
{"msg": "successfully removed memberId
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(orgId : Org1MSP, userId : admin) from role minter"}
```

**`IsInRole`**

This method returns a Boolean value to indicate if a user and token has a specified role. An account ID is formed by creating an SHA-256 hash of the concatenated membership service provider ID (`orgId`) and the user name or email ID (`userId`).

```
Ctx.ERC721Token.IsInRole(role string, accountId string) (bool, error)
```

Parameters:

- `role: string` – The name of the role to check for the specified user. The `mintable` and `burnable` behaviors correspond to the `minter_role_name` and `burner_role_name` properties of the specification file.
- `accountId: number` – The account ID to operate on.

Returns:

- On success, a Boolean value that is true if the role is present for the specified account ID, otherwise false. On error, a non-nil object containing an error message

Example:

```
t.Ctx.ERC721Token.IsInRole(userRole, accountId)

{"result": false}
```

**GetAccountsByRole**

This method returns a list of all account IDs for a specified role.

```
Ctx.ERC721Role.GetAccountsByRole(roleName string) (interface{}, error)
```

Parameters:

- `roleName: string` – The name of the role to search for.

Returns:

- On success, a JSON array of account IDs. On error, a non-nil error object that contains an error message.

Example:

```
t.Ctx.ERC721Role.GetAccountsByRole(userRole)

{
    "accounts": [

"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d"
    ]
}
```

**GetUsersByRole**

This method returns a list of all users for a specified role.

```
Ctx.ERC721Role.GetUsersByRole(roleName string) (interface{}, error)
```

Parameters:

- `roleName: string` – The name of the role to search for.

Returns:

- On success, a JSON array of user objects. Each object contains the user ID and organization ID. On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Role.GetUsersByRole(userRole)
```

```
{
    "Users": [
        {
            "OrgId": "Org1MSP",
            "UserId": "admin"
        }
    ]
}
```

### Methods for Transaction History Management

**`GetAccountTransactionHistory`**
This method returns an array of the transaction history details for a specified account.

```
Ctx.ERC721Account.GetAccountTransactionHistory(accountId string)
(interface{}, error)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, an array of account transaction objects in JSON format:
  - `TransactionId` – The ID of the transaction.
  - `TransactedAccount` – The account with which the transaction took place.
  - `TransactionType` – The type of transaction.
  - `Timestamp` – The time of the transaction.
  - On error, a non-nil error object that contains an error message.

Example:
```
t.Ctx.ERC721Account.GetAccountTransactionHistory(accountId)
```

```
[
     {
        "Timestamp": "2022-04-06T08:31:39Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
        "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
7",
        "TransactionType": "DEBIT"
    }
```

```
    {
        "Timestamp": "2022-04-06T08:16:53Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1",
        "TransactionId":
"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76
392ecf7",
        "TransactionType": "MINT"
    }
]
```

**GetAccountTransactionHistoryWithFilters**

This method returns account transaction history for a specified user, filtered by `PageSize`, `Bookmark`, `startTime` and `endTime`. This method can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC721Account.GetAccountTransactionHistoryWithFilters(accountId
string, filters ...erc721Account.AccountHistoryFilters)
```

Parameters:

- `accountId: string` – ID of the account.
- `filters: string` – An optional parameter. If empty, all records are returned. The `PageSize` property determines the number of records to return. If `PageSize` is 0, the default page size is 20. The `Bookmark` property determines the starting index of the records to return. For more information, see the Hyperledger Fabric documentation. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Returns:

- On success, an array of account transaction objects in JSON format:
  - `TransactionId` – The ID of the transaction.
  - `TransactedAccount` – The account with which the transaction took place.
  - `TransactionType` – The type of transaction.
  - `Timestamp` – The time of the transaction.
  - On error, a non-nil error object that contains an error message.

Example:

```
t.Ctx.ERC721Account.GetAccountTransactionHistoryWithFilters(accountId,
filters...)
```

```
[
    {
        "Timestamp": "2022-04-06T08:31:39Z",
        "TokenId": "monalisa",
        "TransactedAccount":
```

```
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
        "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
7",
        "TransactionType": "DEBIT"
    }
    {
        "Timestamp": "2022-04-06T08:16:53Z",
        "TokenId": "monalisa",
        "TransactedAccount":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
        "TransactionId":
"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf
7",
        "TransactionType": "MINT"
    }
]
```

**GetTransactionById**

This method returns the history of a `Transaction` asset.

```
Ctx.ERC721Transaction.GetTransactionById(trxId string) (interface{}, error)
```

Parameters:

- `trxId: string` – The ID of the transaction asset.

Returns:

- On success, an array of maps of transaction assets. On error, a non-nil error object that contains an error message.

Example:

```
t.Ctx.ERC721Transaction.GetTransactionById(transactionId)
```

```
{
    "History": [
        {
            "IsDelete": "false",
            "Timestamp": "2022-04-06T08:31:39Z",
            "TxId":
"5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac7",
            "Value": {
                "AssetType": "otransaction",
                "Data": "",
                "FromAccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0dfd1",
                "Timestamp": "2022-04-06T08:31:39Z",
                "ToAccountId":
"oaccount~0829f0996744ca9dc8b4e9165a7a8f5db3fdffdc46c96b94f5d625041502cec4",
                "TokenId": "monalisa",
                "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a440e8ac
```

```
7",
                "TransactionType": "TRANSFER",
                "TriggeredByAccountId":
"oaccount~543c2258e351c3e7a40ea59b81e62154d38fbfc9d1b5b79f30ac5e08e7d0d
fd1"
            }
        }
    ],
    "TransactionId":
"otransaction~5a353e02e657c2c8fddce41dd4e7260025fe7beef634ca3351fc366a4
40e8ac7"
}
```

**DeleteHistoricalTransactions**

This method deletes transactions that are older than a specified date from the state database.

```
func (t *Controller) DeleteHistoricalTransactions(referenceTime
string) (interface{}, error)
```

Parameters:

- `referenceTime: string` – Transactions older than the specified time will be deleted.

Returns:

- On success, an array of the deleted transaction IDs and a success message. On error, a non-nil error object that contains an error message.

Example:

```
t.Ctx.ERC721Transaction.DeleteHistoricalTransactions(timestamp)
```

```
{
    "Transactions": [

"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76
392ecf7"
    ],
    "msg": "Successfuly deleted transaction older than
date:2022-04-06T08:17:53Z"
}
```

### Token Behavior Management - Mintable Behavior

**GetMaxMintQuantity**

This method returns the maximum mintable quantity of a token. If the `max_mint_quantity` behavior is not configured in the specification file, then the default value is 0 and an infinite number of tokens can be minted.

```
Ctx.ERC721Token.GetMaxMintQuantity(id string) (float64, error)
```

Parameters:

- `id` – The ID of the token to operate on.

Returns:

- On success, the maximum mintable quantity of the token, in the number data type. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.GetMaxMintQuantity(tokenId);

20000
```

**`GetTotalMintedTokens`**

This method returns the total net number of tokens available in the system for the specified token. The net number of tokens available is the total number of minted tokens minus the number of burned tokens.

```
Ctx.ERC721Token.GetTotalMintedTokens() (map[string]interface{}, error)
```

Parameters:

- none

Returns:

- On success, a map of the total minted tokens, in the number data type, and a success message. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.GetTotalMintedTokens()

{"TotalNetSupply": 5}
```

### Token Behavior Management - Transferable Behavior

**`SafeTransferFrom`**

This method transfers ownership of the specified NFT from the caller to another account. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.
- The caller of the function is the sender.

```
Ctx.ERC721Token.SafeTransferFrom(fromAccountId string, toAccountId string,
tokenAsset interface{}, data ...string) (interface{}, error)
```

Parameters:

- `fromAccountId: string` – The account ID of the sender in the current organization.

- `toAccountId: string` – The account ID of the receiver in the current organization.
- `tokenAsset` – The reference to the token asset to operate on.
- `data: string` – Optional additional information to store in the transaction.

Returns:

- On success, a promise with a success message that includes account details. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.SafeTransferFrom(fromAccountId, toAccountId,
tokenAssetValue.Interface(), data...)

{"msg": "Successfully transferred NFT token: 'monalisa' from Account-
Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad128
8d (Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729d
ba (Org-Id: Org1MSP, User-Id: user1"}
```

**`TransferFrom`**

This method transfers ownership of the specified NFT from a sender account to a receiver account. It is the responsibility of the caller to pass the correct parameters. This method can be called by any user, not only the token owner. This method includes the following validations:

- The token exists and is not burned.
- The sender account and receiver account exist and are not the same account.
- The sender account owns the token.

```
Ctx.ERC721Token.TransferFrom(fromAccountId string, toAccountId string,
tokenAsset interface{}) (interface{}, error)
```

Parameters:

- `fromAccountId: string` – The account ID of the sender in the current organization.
- `toAccountId: string` – The account ID of the receiver in the current organization.
- `tokenAsset` – The reference to the token asset to operate on.

Returns:

- On success, a promise with a success message that includes account details. Account IDs have the prefix `oaccount~`. On error, a non-nil object with an error message.

\Example:

```
t.Ctx.ERC721Token.TransferFrom(fromAccountId, toAccountId,
tokenAssetValue.Interface())

{"msg": "Successfully transferred NFT token: 'monalisa' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin) to Account-Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1"}
```

### Token Behavior Management - Burnable Behavior

**`Burn`**

This method deactivates, or burns, the specified NFT from the caller's account. The caller of this method must have an account. A token cannot be burned unless the token specification file includes the `burnable` behavior. If no `burner_role_name` property is specified in the `roles` section of the specification file, then the owner of the token can burn the token. If a `burner_role_name` property is specified in the `roles` section, then the user assigned the burner role who is also the minter (creator) of the token can burn the token. The `burn` method is part of the `ERC721Token` package, which you access via the receiver of the `Ctx` struct.

```
Ctx.Token.Burn(tokenAsset interface{}) (interface{}, error)
```

Parameters:

- `tokenAsset` – The reference to the token asset to operate on.

Returns:

- On success, a promise with a success message that includes account details. On error, a non-nil object with an error message.

Example:

```
t.Ctx.ERC721Token.Burn(tokenAssetValue.Interface())

{"msg": "Successfully burned NFT token: 'monalisa' from Account-Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin"}
```

## Go Methods for ERC-721 NFT Locking

Blockchain App Builder automatically generates methods that you can use to lock non-fungible tokens that use the extended ERC-721 standard.

A locked token cannot be burned or transferred to other users. All other properties, such as the token's state, owner, and history are preserved. You can use the NFT locking functionality when transferring a token to another blockchain network, such as Ethereum or Polygon.

Before you can lock NFTs, you must assign the vault manager role to a user. The vault manager is a special type of role, a `TokenSys` role. `TokenSys` roles are different from asset-based roles such as burner, minter, and notary, and from administrative roles such as `Token Admin` and `Org Admin`. Currently Blockchain App Builder supports the `vault TokenSys` role. The single user who has the `vault` role for a chaincode is the vault manager of the chaincode, and can manage locked NFTs.

The typical flow for using the NFT locking functionality follows these steps.

- Create a non-fungible token that has the lockable behavior.
- Use the `AddTokenSysRole` method to give the `vault` role to a user, the vault manager.
- Call the `LockNFT` method to lock a non-fungible token, specified by the token ID.

### TokenSys Role Management Methods

**`AddTokenSysRole`**
This method adds a `TokenSys` role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenSysRole(role string, orgId string, userId
string) (interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.AddTokenSysRole",
"TOKEN", map[string]string{"accountId": accountId})
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Token.AddTokenSysRoleMember(role, accountId)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to give to the user.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully added role 'vault' to Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1)"
}
```

**IsInTokenSysRole**

This method returns a Boolean value to indicate if a user has a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) IsInTokenSysRole(orgId string, userId string, role
string) (interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.IsInTokenSysRole", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Token.IsInTokenSysRoleMember(role, accountId)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to check.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1) has vault role"
}
```

**RemoveTokenSysRole**

This method removes a `TokenSys` role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) RemoveTokenSysRole(role string, orgId string, userId
string) (interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.RemoveTokenSysRole",
```

```
"TOKEN", map[string]string{"accountId": accountId})
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Token.RemoveTokenSysRoleMember(role, accountId)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to remove.
- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully removed role 'vault' from Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1)"
}
```

**`TransferTokenSysRole`**

This method transfers a `TokenSys` role from a user to another user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TransferTokenSysRole(role string, fromOrgId
string, fromUserId string, toOrgId string, toUserId string)
(interface{}, error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.TransferTokenSysRole",
 "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    fromAccountId, err :=
t.Ctx.ERC721Account.GenerateAccountId(fromOrgId, fromUserId)
    if err != nil {
        return nil, fmt.Errorf("error in TransferTokenSysRole. Error:
%s", err)
    }
    toAccountId, err := t.Ctx.ERC721Account.GenerateAccountId(toOrgId,
toUserId)
    if err != nil {
        return nil, fmt.Errorf("error in TransferTokenSysRole. Error:
```

```
%s", err)
    }
    return t.Ctx.ERC721Token.TransferTokenSysRole(role, fromAccountId,
toAccountId)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to transfer.
- `fromOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role from.
- `fromUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role from.
- `toOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role to.
- `toUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role to.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully transfered role 'vault' from Account Id:
ouaccount~f4e311528f03fffa7810753d643f66289ff6c9080fcf839902f28a1d3aff1789
(Org-Id: Org1MSP, User-Id: user1) to Account Id:
ouaccount~ae5be2ae8f98d6d32f5d02b43877d987114e7937c7bacbc30390dcce09996a19
(Org-Id: Org1MSP, User-Id: user2)"
}
```

**GetAccountsByTokenSysRole**

This method returns a list of all account IDs for a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetAccountsByTokenSysRole(role string) (interface{},
error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetAccountsByTokenSysRole",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Token.GetAccountsByTokenSysRole(role)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "accountIds": [

"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03
eba"
    ]
}
```

**GetUsersByTokenSysRole**

This method returns user information for all users with a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetUsersByTokenSysRole(role string) (interface{},
error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetUsersByTokenSysRole
", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Token.GetUsersByTokenSysRole(role)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "Users":[
      {

"accountId":"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb8
3c874c2caf03eba",
         "orgId":"Org1MSP",
         "userId":"user1"
      }
```

```
   ]
}
```

### NFT Locking Methods

**`LockNFT`**
This method locks a specified non-fungible token. To lock a token, there must be a user with the `TokenSys vault` role, who acts as the vault manager. This method can be called only by the owner of the token.

```
func (t *Controller) LockNFT(tokenId string) (interface{}, error) {
    return t.Ctx.ERC721Token.LockNFT(tokenId)
}
```

Parameters:

- `tokenID: string` – The ID of the token to lock.

Returns:

- On success, a JSON representation of the token object.

Return Value Example:

```
{
   "AssetType":"otoken",
   "Behavior":[
      "indivisible",
      "singleton",
      "mintable",
      "transferable",
      "lockable",
      "burnable",
      "roles"
   ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
   "CreationDate":"2023-10-20T12:39:50Z",
   "IsBurned":false,
   "IsLocked":true,
   "Mintable":{
      "Max_mint_quantity":20000
   },
   "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
   "Price":120,
   "Roles":{
      "minter_role_name":"minter"
   },
   "Symbol":"ART",
```

```
   "TokenDesc":"",
   "TokenId":"token1",
   "TokenMetadata":{
      "Description":"",
      "Image":"",
      "Painter_name":"",
      "Painting_name":""
   },
   "TokenName":"artcollection",
   "TokenStandard":"erc721+",
   "TokenType":"nonfungible",
   "TokenUnit":"whole",
   "TokenUri":"token1.example.com"
}
```

**IsNFTLocked**

This method returns a Boolean value to indicate if a specified token is locked. This method can be called only by the token owner, the vault manager (the user with the `TokenSys vault` role), or a `Token Admin` of the chaincode.

```
func (t *Controller) IsNFTLocked(tokenId string) (interface{}, error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.IsNFTLocked",
"TOKEN", map[string]string{"tokenId": tokenId})
    if err != nil && !auth {
        isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC721Token.IsCallerTokenSysRoleHolder(constants.Vault)
        if error != nil {
            return nil, error
        }
        if !isCallerTokenSysRoleHolder {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
        }
    }
    return t.Ctx.ERC721Token.IsNFTLocked(tokenId)
}
```

Parameters:

- `tokenID: string` – The ID of the token.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "isNFTLocked":true
}
```

**GetAllLockedNFTs**

This method returns a list of all locked non-fungible tokens. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
func (t *Controller) GetAllLockedNFTs() (interface{}, error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetAllLockedNFTs", "TOKEN")
    if err != nil && !auth {
        isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC721Token.IsCallerTokenSysRoleHolder(constants.Vault)
        if error != nil {
            return nil, error
        }
        if !isCallerTokenSysRoleHolder {
            return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
        }
    }
    return t.Ctx.ERC721Token.GetAllLockedNFTs()
}
```

Parameters:

- None

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:

```
[
   {
      "key":"token1",
      "valueJson":{
         "AssetType":"otoken",
         "Behavior":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "lockable",
            "burnable",
            "roles"
         ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "CreationDate":"2023-10-20T12:39:50Z",
         "IsBurned":false,
         "IsLocked":true,
         "Mintable":{
```

```
            "Max_mint_quantity":20000
         },
         "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff
1b6a7733463",
         "Price":120,
         "Roles":{
            "minter_role_name":"minter"
         },
         "Symbol":"ART",
         "TokenDesc":"",
         "TokenId":"token1",
         "TokenMetadata":{
            "Description":"",
            "Image":"",
            "Painter_name":"",
            "Painting_name":""
         },
         "TokenName":"artcollection",
         "TokenStandard":"erc721+",
         "TokenType":"nonfungible",
         "TokenUnit":"whole",
         "TokenUri":"token1.example.com"
      }
   }
]
```

**GetAllLockedNFTsByOrg**

This method returns a list of all locked non-fungible tokens for a specified organization and optionally a specified user. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
func (t *Controller) GetLockedNFTsByOrg(orgId string,
userId ...string) (interface{}, error) {
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721TOKEN.GetLockedNFTsByOrg",
"TOKEN")
    if err != nil && !auth {
        isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC721Token.IsCallerTokenSysRoleHolder(constants.Vault)
        if error != nil {
            return nil, error
        }
        if !isCallerTokenSysRoleHolder {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
        }
    }
    return t.Ctx.ERC721Token.GetLockedNFTsByOrg(orgId, userId...)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user (optional).

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:


```
[
   {
      "key":"token1",
      "valueJson":{
         "AssetType":"otoken",
         "Behavior":[
            "indivisible",
            "singleton",
            "mintable",
            "transferable",
            "lockable",
            "burnable",
            "roles"
         ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "CreationDate":"2023-10-20T12:39:50Z",
         "IsBurned":false,
         "IsLocked":true,
         "Mintable":{
            "Max_mint_quantity":20000
         },
         "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
         "Price":120,
         "Roles":{
            "minter_role_name":"minter"
         },
         "Symbol":"ART",
         "TokenDesc":"",
         "TokenId":"token1",
         "TokenMetadata":{
            "Description":"",
            "Image":"",
            "Painter_name":"",
            "Painting_name":""
         },
         "TokenName":"artcollection",
         "TokenStandard":"erc721+",
         "TokenType":"nonfungible",
```

```
            "TokenUnit":"whole",
            "TokenUri":"token1.example.com"
        }
    }
]
```

## Go Methods for ERC-721 Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for tokens that use the extended ERC-721 standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, and transferring tokens. Additionally, other users cannot transfer tokens to a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.

- Automatically Generated Account Status Methods
- Account Status SDK Methods

### Automatically Generated Account Status Methods

**`GetAccountStatus`**
This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
func (t *Controller) GetAccountStatus(orgId string, userId string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721AccountStatus.Get",
"TOKEN", map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      accountStatus, err :=
t.Ctx.ERC721AccountStatus.GetAccountStatus(accountId)
      if err != nil {
            return
t.Ctx.ERC721AccountStatus.GetDefaultAccountStatus(accountId)
      }
```

```
      return accountStatus, nil
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
func (t *Controller) GetAccountStatusHistory(orgId string, userId string)
(interface{}, error) {
      accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
      }
      _, err = t.Ctx.ERC721Account.GetAccount(accountId)
      if err != nil {
            return nil, fmt.Errorf("error in GetAccountStatusHistory: %s",
err)
      }
      auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721AccountStatus.Get", "TOKEN",
map[string]string{"accountId": accountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      statusId, err :=
t.Ctx.ERC721AccountStatus.GenerateAccountStatusId(accountId)
```

```
      if err != nil {
            return nil, err
      }
      accountStatusHistory, err :=
t.Ctx.ERC721AccountStatus.History(statusId)
      if err != nil {
            return []map[string]interface{}{}, nil
      }
      return accountStatusHistory, nil
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, the account status history in JSON format.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
```

```
79d5e96d7"
    }
  }
]
```

**ActivateAccount**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode. Deleted accounts cannot be activated. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
func (t *Controller) ActivateAccount(orgId string, userId string)
(interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating accountId of
(Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721AccountStatus.ActivateAccount",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Account.ActivateAccount(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**SuspendAccount**

This method suspends a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
func (t *Controller) SuspendAccount(orgId string, userId string)
(interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId,
userId)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721AccountStatus.SuspendAccount
", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Account.SuspendAccount(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "suspended"
}
```

**DeleteAccount**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot

be changed to any other state. To delete an account, the account balance must be zero.

```
func (t *Controller) DeleteAccount(orgId string, userId string)
(interface{}, error) {
    accountId, err := t.Ctx.ERC721Account.GenerateAccountId(orgId, userId)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating accountId of
(Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC721Auth.CheckAuthorization("ERC721AccountStatus.DeleteAccount",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC721Account.DeleteAccount(accountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "deleted"
}
```

### Account Status SDK Methods

**`GetDefaultAccountStatus`**
This method gets the current status of a token account, with the status as `active` for any account that does not have account status stored in the ledger (because the account was created prior to the account status functionality).

```
Ctx.ERC721AccountStatus.GetDefaultAccountStatus(accountId string)
(NFTAccountStatus, error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
      "AssetType": "oaccountStatus",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "Status": "active"
}
```

**`GetAccountStatus`**
This method gets the current status of the token account.

```
Ctx.ERC721AccountStatus.GetAccountStatus(accountId string)
(NFTAccountStatus, error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**
This method gets the history of the account status.

```
Ctx.ERC721AccountStatus.History(statusId string) (interface{}, error)
```

Parameters:

- `statusId: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:


```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7"
    }
  }
]
```

**ActivateAccount**

This method activates a token account. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
Ctx.ERC721Account.ActivateAccount(accountId string) (interface{},
error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "active"
}
```

**SuspendAccount**

This method suspends a token account.

```
Ctx.ERC721Account.SuspendAccount(accountId string) (interface{}, error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
```

```
9c1",
    "Status": "suspended"
}
```

**DeleteAccount**
This method deletes a token account.

```
Ctx.ERC721Account.DeleteAccount(accountId string) (interface{}, error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "deleted"
}
```

# ERC-1155

Blockchain App Builder supports an extended version of the ERC-1155 standard to work with fungible and non-fungible tokens.

- Input Specification File for ERC-1155
- ERC-1155 Tokenization Flow
- Scaffolded TypeScript Token Project for ERC-1155
- Scaffolded Go Token Project for ERC-1155

## Input Specification File for ERC-1155

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

You can define standard assets and both fungible and non-fungible token assets that are based on the ERC-1155 standard in the same specification file. You cannot define token assets based on more than one standard in the same specification file.

For information on including standard assets in the specification file, see Input Specification File.

The following sample specification files for ERC-1155 token assets are available in the Blockchain App Builder package:

- `NFTArtCollectionMarketplacewithERC1155-TypeScript.yml`
- `FractionalNFTinRealEstate-TypeScript.yml`

In addition to the standard properties and sections, token assets support the `behavior` and `anatomy` sections in the specification file. In addition, non-fungible token assets support the `metadata` section. The following example shows the structure of a specification file for two ERC-1155 token assets, a whole non-fungible token and a fractional fungible token:

```
assets:
    - name: ArtCollection #Asset name
      type: token #Asset type
      standard: erc1155+   # Token standard

      anatomy:
          type: nonfungible # Token type
          unit: whole  #Token unit

      behavior:
        - indivisible
        - mintable:
            max_mint_quantity: 20000
        - transferable
        - burnable
        - lockable
        - roles:
            minter_role_name: minter

      properties:  # Custom asset attributes for non-fungible token

          - name: price # Custom asset attribute to set the price of a
non-fungible token in the marketplace
            type: number

          - name: on_sale_flag # Custom asset attribute maintains non-
fungible token selling status in the marketplace
            type: boolean

      metadata: #  Use this section to maintain the metadata on the
blockchain. Only the user creating the non-fungible token can assign
metadata attribute values, which cannot be updated later.

          - name: painting_name
            type: string

          - name: description
            type: string

          - name: image
            type: string

          - name: painter_name
```

```
            type: string

    - name: Loyalty # Asset name
      type: token  # Asset type
      standard: erc1155+   # Token standard

      anatomy:
          type: fungible # Token type
          unit: fractional # Token unit

      behavior: # Token behaviors
          - divisible:
                decimal: 2
          - mintable:
                max_mint_quantity: 10000
          - transferable
          - burnable
          - roles:
                minter_role_name: minter

      properties:
          - name: currency_name # Custom attribute to represent the token in
a specific currency.
            type: string

          - name: token_to_currency_ratio # Custom attribute to specify the
token to currency ratio.
            type: number
```

The following example shows the structure of a specification file for a fractional non-fungible token:

```
- name: RealEstateProperty #Asset name
      type: token #Asset type
      standard: erc1155+ # Token standard

      anatomy:
          type: nonfungible # Token type
          unit: fractional #Token unit

      behavior:
        - divisible:
        - mintable:
        - transferable
        - roles:
            minter_role_name: minter

      properties: # Custom asset attributes for non-fungible token.

          - name: propertySellingPrice # Custom asset attribute to set the
real estate property selling price
            type: number

          - name: propertyRentingPrice # Custom asset attribute maintains
```

```
the renting amount for the real estate property
            type: number

      metadata: # To maintain the metadata on-chain, this tag will be
used. Users won't be able to update the metadata attribute values.

          - name: propertyType
            type: string

          - name: propertyName
            type: string

          - name: propertyAddress
            type: string

          - name: propertyImage
            type: string
```

**Table 7-7 Parameter Descriptions and Examples for an ERC-1155 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`type:`** | You must specify `type: token` in the `assets` section. | `assets:`<br>`- name: ArtCollection #Asset name`<br>`type: token #Asset type` |
| **`standard:`** | The `standard` property is mandatory for ERC-1155 tokens. It represents the token standard to follow during chaincode generation. | `standard: erc1155+ # Token standard` |

**Table 7-7 (Cont.) Parameter Descriptions and Examples for an ERC-1155 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`anatomy:`** | The `anatomy` section has two mandatory parameters:<br>• `type: nonfungible` or `type: fungible`<br>A non-fungible token is unique.<br>• `unit: whole` or `unit: fractional`<br>A whole token cannot be subdivided into smaller fractional units. A fractional token can be subdivided into smaller units, or shares, based on a specified number of decimal places. | `anatomy:`<br>`  type: nonfungible # Token type`<br>`  unit: whole #Token unit` |

**Table 7-7 (Cont.) Parameter Descriptions and Examples for an ERC-1155 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| **`behavior:`** | This section describes the capabilities and restrictions of the token. The `mintable`, `transferable` behaviors are mandatory for all tokens. The `indivisible` behavior is mandatory for whole non-fungible tokens.<br>• `indivisible`: This behavior supports a restriction so that whole tokens cannot be subdivided into fractions.<br>• `divisible`: This behavior describes how tokens can be subdivided. The `decimal` parameter specifies the number of decimal places that can be used. The smallest fraction possible with the number of decimal places is the smallest unit of the token that can be owned. If the `decimal` parameter is not specified, the default is zero decimal places.<br>• `mintable`: This mandatory behavior supports minting new token instances. The optional | `behavior:`<br>`- indivisible`<br>`- mintable:`<br>`max_mint_quantity: 20000`<br>`- transferable`<br>`- burnable`<br>`- lockable`<br>`- roles:`<br>`minter_role_name: minter` |

**Table 7-7 (Cont.) Parameter Descriptions and Examples for an ERC-1155 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | `max_mint_quantity` parameter specifies the total number of tokens that can be minted. If you do not specify the `max_mint_quantity` parameter, any number of tokens can be minted.<br>• `transferable`: This mandatory behavior supports transferring ownership of tokens.<br>• `burnable`: This behavior supports deactivating, or burning, tokens. Burning does not delete a token but instead places it in a permanent state where it cannot be used. Burning is not reversible.<br>• `lockable`: This behavior is optional and is supported only by non-fungible tokens. This behavior allows the token owner to lock a non-fungible token. A locked token cannot be transferred to or burned by any other users.<br>• `roles`: This behavior restricts token behaviors to users with specific roles. | |

**Table 7-7 (Cont.) Parameter Descriptions and Examples for an ERC-1155 Token Specification File**

| Entry | Description | Examples |
|---|---|---|
| | Currently two roles are available: `minter_role_name` and `burner_role_name`. If you do not specify roles, then any user can act as a minter or burner. For example, if the burner role is not specified, any account user implicitly has the burner role. If the burner role is specified, then during the token setup process, the `Token Admin` user must assign the burner role to other users explicitly. | |
| **`metadata:`** | The `metadata` property is optional and is supported only by non-fungible tokens. This property specifies metadata information, which is stored on the blockchain, for a non-fungible token. Metadata attribute values can be assigned only by the token owner who mints the token, and cannot be updated after the token is minted.<br>In the example, `name` is the name of the metadata attribute and `type` is the type of value that the attribute has. | `metadata:`<br>`painting_name`<br>`- name:`<br>`type: string`<br>`description`<br>`- name:`<br>`type: string`<br>`- name: image`<br>`type: string`<br>`painter_name`<br>`- name:`<br>`type: string` |

### Limitations

Blockchain App Builder provides partial support for the ERC-1155 standard. Currently, the following ERC-1155 events and methods are not supported.

Events:

- `TransferSingle`
- `TransferBatch`
- `ApprovalForAll`
- `URI`

Methods:

- `safeTransferFrom`
- `balanceOf`
- `setApprovalForAll`
- `isApprovedForAll`

## ERC-1155 Tokenization Flow

After you deploy an ERC-1155 token project, token administrators and token owners follow a typical flow for creating tokens and completing lifecycle operations.

When you deploy a token project, the users in the list passed to the initialization method become token administrators of the chaincode. After deployment, the typical flow for creating tokens and completing lifecycle operations follows these steps:

Token administrator operations:

- Create user accounts for anyone who will possess tokens or complete token-related operations.
- For each user account, create token accounts. Users can have multiple fungible token accounts, but only one non-fungible token (NFT) account. Token administrators can use the `createAccount` method to create user and token accounts simultaneously instead of separately.
- Create fungible tokens, as needed. When you initialize a fungible token, you can assign the associated metadata and behaviors to the token.
- For fungible tokens, associate the fungible token accounts of users to specific fungible tokens.
- Add minter and burner roles to the token accounts of users as needed.

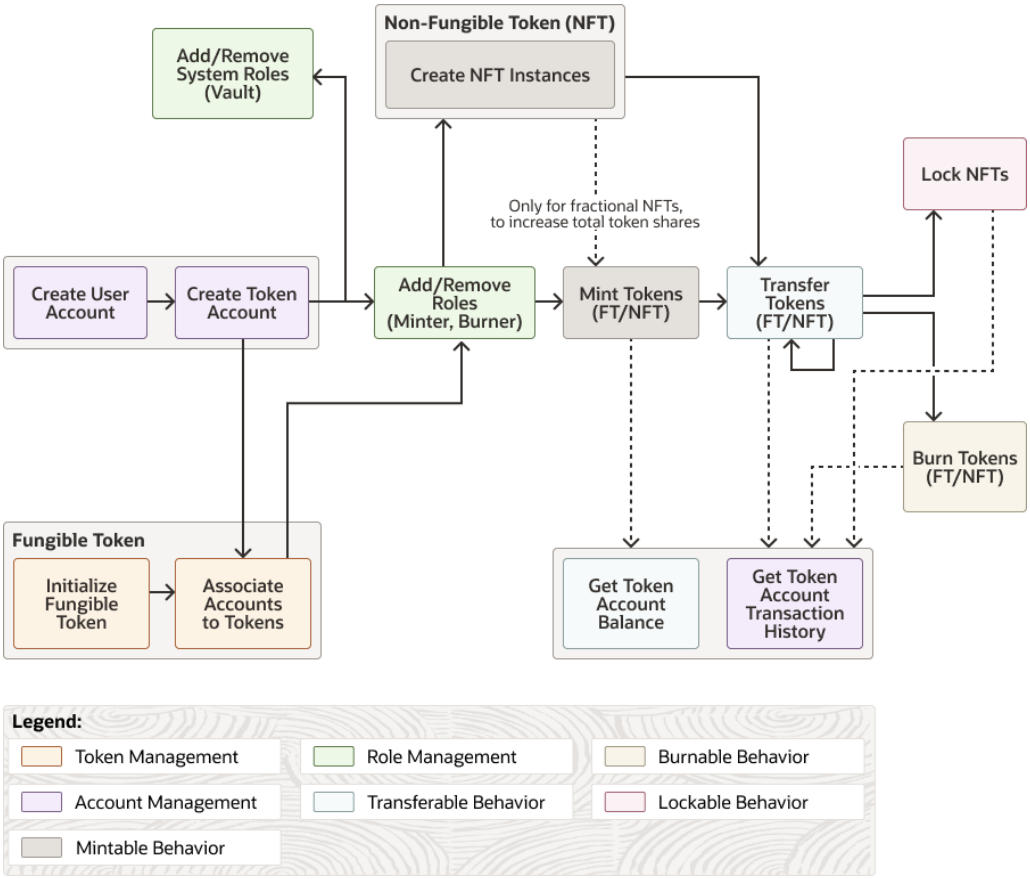
Token owner operations:

- Users who have the minter role for a specific token can create (mint) NFTs or fungible tokens.
- Users can transfer tokens between accounts, and check account balances.
- Users who have the burner role for a specific token can destroy (burn) NFTs or fungible tokens.

Vault manager operations:

- The user who has the vault role can lock NFTs. A locked NFT cannot be burned or transferred to other users.

The following diagram shows the overall process flow for an ERC-1155 tokenization scenario.




The following table summarizes the methods that are automatically generated when you scaffold an ERC-1155 token project.

| Method Category | Auto-generated Method (TypeScript) | Auto-generated Method (Go) | Called By | Description |
|---|---|---|---|---|
| Admin Management | `init` | `Init` | Admin | Initializes the token chaincode |
| | `isTokenAdmin` | `IsTokenAdmin` | Admin | Returns true if the caller is an Admin |
| | `addTokenAdmin` | `AddTokenAdmin` | Admin | Adds an Admin |
| | `removeTokenAdmin` | `RemoveTokenAdmin` | Admin | Removes an Admin |
| | `getAllTokenAdmins` | `GetAllTokenAdmins` | Admin | Returns all Admins |
| Account Management | `createAccount` | `CreateAccount` | Admin | Creates a user account and token accounts |

| Method Category | Auto-generated Method (TypeScript) | Auto-generated Method (Go) | Called By | Description |
|---|---|---|---|---|
| | createUserAccount | CreateUserAccount | Admin | Creates a user account |
| | createTokenAccount | CreateTokenAccount | Admin | Creates a token account |
| | associateFungibleTokenAccount | AssociateFungibleTokenAccount | Admin | Associates a fungible token account with a fungible token |
| | getAccountHistory | GetAccountHistory | Admin / Account Owner | Returns history for a token account |
| | getAccountTransactionHistory | GetAccountTransactionHistory | Admin / Account Owner | Returns transaction history for an account |
| | getAccount | GetAccount | Admin / Account Owner | Returns details for a token account |
| | getAllAccounts | GetAllAccounts | Admin | Returns details for all user accounts |
| | getAccountDetailsByUser | GetAccountDetailsByUser | Admin / Account Owner | Returns details for a user account and all associated tokens |
| | getUserByAccountId | GetUserByAccountId | Any user | Returns user details for an account ID |
| Role Management | addRole | AddRole | Admin | Adds a role to a user and token |
| | isInRole | IsInRole | Admin / Account Owner | Returns whether a user has a specified role for a token |
| | removeRole | RemoveRole | Admin | Removes a role from a user and token |
| | getAccountsByRole | GetAccountsByRole | Admin | Returns account IDs for a specified role and token |
| | getUsersByRole | GetUsersByRole | Admin | Returns a list of users for a specified role and token |
| Mintable Behavior | mintBatch | MintBatch | Users with the minter role | Mints multiple tokens |
| Transferable Behavior | batchTransferFrom | BatchTransferFrom | Any user | Transfers tokens between users |

| Method Category | Auto-generated Method (TypeScript) | Auto-generated Method (Go) | Called By | Description |
|---|---|---|---|---|
| | safeBatchTransferFrom | SafeBatchTransferFrom | Any user | Transfers tokens between the method caller and another user |
| | balanceOfBatch | BalanceOfBatch | Admin / Account Owner | Returns token account balances for multiple users and tokens |
| | exchangeToken | ExchangeToken | Account Owner | Exchanges tokens between token accounts |
| Burnable Behavior | burnBatch | BurnBatch | Users with the burner role | Burns tokens |
| Token Management | create<TokenName>Token | Create<TokenName>Token | Admin (fungible tokens) / Users with the minter role (NFTs) | Creates tokens |
| | update<TokenName>Token | Update<TokenName>Token | Admin (fungible tokens) / Token Owner (NFTs) | Updates tokens |
| | getTokenHistory | GetTokenHistory | Any user | Returns the history of a token |
| | getTransactionById | GetTransactionById | Any user | Returns the details of a specified transaction |
| | deleteHistoricalTransactions | DeleteHistoricalTransactions | Admin | Deletes transactions before a specified time |
| | getAllTokens | GetAllTokens | Admin | Returns all token assets |
| | getTokenById | GetTokenById | Admin / Token Owner | Returns a token |
| | getAllTokensByUser | GetAllTokensByUser | Admin / Account Owner | Returns all tokens owned by a specified user |
| | ownerOf | OwnerOf | Any user | Returns the user details of the owner of a specified token |
| | URI | URI | Any user | Returns the URI of a specified token |
| | name | Name | Any user | Returns the name of a specified token |
| | totalSupply | TotalSupply | Admin | Returns the number of minted tokens for a specified token |

| Method Category | Auto-generated Method (TypeScript) | Auto-generated Method (Go) | Called By | Description |
|---|---|---|---|---|
| | totalNetSupply | TotalNetSupply | Admin | Returns the number of minted tokens minus the number of burned tokens for a specified token |
| | getTokensByName | getTokensByName | Admin | Returns all token assets for a specified token name |
| | getTokenDecimal | getTokenDecimal | Admin | Returns the number of decimal places for a specified token |

## Scaffolded TypeScript Token Project for ERC-1155

Blockchain App Builder takes the input from your token specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates token lifecycle classes and functions, including CRUD and non-CRUD methods. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to tokens, see Scaffolded TypeScript Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated Token Methods
- SDK Methods

### Model

Every tokenized model class extends the `OchainModel` class. Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class. The following model shows a whole non-fungible token.

```
import * as yup from "yup";
import { Id, Mandatory, Validate, Default, Embedded, Derived, ReadOnly }
from "../../lib/decorators";
import { OchainModel } from "../../lib/ochain-model";
import { STRATEGY } from "../../lib/utils";
import { EmbeddedModel } from "../../lib/ochain-embedded-model";

export class ArtCollectionMetadata extends
EmbeddedModel<ArtCollectionMetadata> {
  @Validate(yup.string())
```

```
  public painting_name: string;

  @Validate(yup.string())
  public description: string;

  @Validate(yup.string())
  public image: string;

  @Validate(yup.string())
  public painter_name: string;

}

@Id("tokenId")
export class ArtCollection extends OchainModel<ArtCollection> {
  public readonly assetType = "otoken";

  @Mandatory()
  @Validate(
    yup
      .string()
      .required()
      .matches(/^[A-Za-z0-9][A-Za-z0-9_-]*$/)
      .max(16)
  )
  public tokenId: string;

  @ReadOnly("artcollection")
  public tokenName: string;

  @Validate(yup.string().trim().max(256))
  public tokenDesc: string;


  @ReadOnly("erc1155+")
  public tokenStandard: string;

  @ReadOnly("nonfungible")
  public tokenType: string;

  @ReadOnly("whole")
  public tokenUnit: string;


@ReadOnly(["indivisible","singleton","mintable","transferable","burnabl
e","roles"])
  public behaviors: string[];

  @ReadOnly({ minter_role_name: "minter" })
  public roles: object;

  @ReadOnly({ max_mint_quantity: 20000 })
  public mintable: object;

  @Validate(yup.string())
```

```
  public owner: string;

  @Validate(yup.string())
  public createdBy: string;

  @Validate(yup.string())
  public transferredBy: string;

  @Validate(yup.string())
  public creationDate: string;

  @Validate(yup.string())
  public transferredDate: string;

  @Validate(yup.bool())
  public isBurned: boolean;

  @Validate(yup.string())
  public burnedBy: string;

  @Validate(yup.string())
  public burnedDate: string;

  @Mandatory()
  @Validate(yup.string().required().max(2000))
  public tokenUri: string;

  @Embedded(ArtCollectionMetadata)
  public tokenMetadata: ArtCollectionMetadata;

  @Validate(yup.number())
  public price: number;

  @Validate(yup.boolean())
  public on_sale_flag: boolean;

}

@Id("tokenId")
export class Loyalty extends OchainModel<Loyalty> {
  public readonly assetType = "otoken";

  @Mandatory()
  @Validate(
    yup
      .string()
      .required()
      .matches(/^[A-Za-z0-9][A-Za-z0-9_-]*$/)
      .max(16)
  )
  public tokenId: string;

  @ReadOnly("loyalty")
  public tokenName: string;
```

```
  @Validate(yup.string().trim().max(256))
  public tokenDesc: string;

  @ReadOnly("erc1155+")
  public tokenStandard: string;

  @ReadOnly("fungible")
  public tokenType: string;

  @ReadOnly("fractional")
  public tokenUnit: string;

  @ReadOnly(["divisible","mintable","transferable","burnable","roles"])
  public behaviors: string[];

  @ReadOnly({ minter_role_name: "minter" })
  public roles: object;

  @ReadOnly({ max_mint_quantity: 10000 })
  public mintable: object;

  @ReadOnly({ decimal: 2 })
  public divisible: object;

  @Validate(yup.string())
  public currency_name: string;

  @Validate(yup.number())
  public token_to_currency_ratio: number;

}
```

The following model shows a fractional non-fungible token.

```
export class RealEstatePropertyMetadata extends
EmbeddedModel<RealEstatePropertyMetadata> {
  @Validate(yup.string())
  public propertyType: string;

  @Validate(yup.string())
  public propertyName: string;

  @Validate(yup.string())
  public propertyAddress: string;

  @Validate(yup.string())
  public propertyImage: string;

}

@Id("tokenId")
export class RealEstateProperty extends
OchainModel<RealEstateProperty> {
  public readonly assetType = "otoken";
```

```
@Mandatory()
@Validate(
  yup
    .string()
    .required()
    .matches(/^[A-Za-z0-9][A-Za-z0-9_-]*$/)
    .max(16)
)
public tokenId: string;

@ReadOnly("realestateproperty")
public tokenName: string;

@Validate(yup.string().trim().max(256))
public tokenDesc: string;

@ReadOnly("erc1155+")
public tokenStandard: string;

@ReadOnly("nonfungible")
public tokenType: string;

@ReadOnly("fractional")
public tokenUnit: string;

@ReadOnly(["divisible","mintable","transferable","roles"])
public behaviors: string[];

@ReadOnly({ minter_role_name: "minter" })
public roles: object;

@ReadOnly({ max_mint_quantity: 0 })
public mintable: object;

@Validate(yup.number().positive())
public quantity: number;

@Validate(yup.string())
public createdBy: string;

@Validate(yup.string())
public creationDate: string;

@ReadOnly({ decimal: 0 })
public divisible: object;

@Validate(yup.bool())
public isBurned: boolean;

@Mandatory()
@Validate(yup.string().required().max(2000))
public tokenUri: string;

@Embedded(RealEstatePropertyMetadata)
```

```
  public tokenMetadata: RealEstatePropertyMetadata;

  @Validate(yup.number())
  public propertySellingPrice: number;

  @Validate(yup.number())
  public propertyRentingPrice: number;

}
```

### Controller

The main controller class extends the `OchainController` class. There is only one main controller.

```
export class DigiCurrCCController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

### Automatically Generated Token Methods

Blockchain App Builder automatically generates methods to support tokens and token life cycles. You can use these methods to initialize tokens, manage roles and accounts, and complete other token lifecycle tasks without any additional coding. Controller methods must have a `@Validator(...params)` decorator to be invokable.

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

#### Methods for Access Control Management

**isTokenAdmin**

This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAccountDetailsByUser(orgId: string, userId: string) {
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.getAccountDetailsByUs
er", "TOKEN", {
    accountId: userAccountId,
  });
  return await this.Ctx.ERC1155Account.getAccountDetailsByUser(orgId,
userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

Return Value Example:

```
{"result": true}
```

**addTokenAdmin**

This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
public async addTokenAdmin(orgId: string, userId: string) {
  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ADMIN.addAdmin",
"TOKEN");
  return await this.Ctx.ERC1155Admin.addAdmin(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully added Admin (OrgId: appDev, UserId: user1)"}
```

**removeTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. You cannot remove yourself as a `Token Admin`.

```
@Validator(yup.string(), yup.string())
public async removeTokenAdmin(orgId: string, userId: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ADMIN.removeAdmin",
"TOKEN");
  return await this.Ctx.ERC1155Admin.removeAdmin(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg": "Successfully removed Admin (OrgId: appDev, UserId: user1)"}
```

**getAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async getAllTokenAdmins() {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ADMIN.getAllAdmins",
"TOKEN");
  return await this.Ctx.ERC1155Admin.getAllAdmins();
}
```

Parameters:

- none

Returns:

- On success, an `admins` array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{
  "admins": [
    {
      "orgId": "appdev",
      "userId": "user2"
    },
    {
      "orgId": "appdev",
      "userId": "user1"
    }
  ]
}
```

### Methods for Token Configuration Management

**`init`**
This method is called when the chaincode is instantiated. Every `Token Admin` is identified by the `userId` and `orgId` information in the `adminList` parameter. The `userId` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `orgId` is the membership service provider (MSP) ID of the user in the current network organization. The `adminList` parameter is mandatory the first time you deploy the chaincode. If you are upgrading the chaincode, pass an empty list (`[]`). If you are the user who initially deployed the chaincode, you can also specify new admins in the `adminList` parameter when you are upgrading the chaincode. Any other information in the `adminList` parameter is ignored during upgrades.

```
@Validator(yup.array().of(yup.object()).nullable())
public async init(adminList: ERC1155TokenAdminAsset[]) {
  await this.Ctx.ERC1155Admin.initAdmin(adminList);
  await this.Ctx.ERC1155Token.saveClassInfo(<1st NFT Token Name>);
  await this.Ctx.ERC1155Token.saveClassInfo(<2nd NFT Token Name>);
  .
  .
  await this.Ctx.ERC1155Token.saveClassInfo(<nth NFT Token Name>);
  // await this.Ctx.ERC1155Token.saveDeleteTransactionInfo();
  return;
}
```

Parameters:

- `adminList array` – An array of `{orgId, userId}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

**`create<Token Name>Token`**
This method creates tokens. Every token that is defined has its own create method. For fungible tokens, this method can be called only by a `Token Admin` of the chaincode. For non-fungible tokens, if the minter role is defined in the specification file, any user with the minter

role can call this method to create an NFT. If the minter role is not defined, any user can use this method to create (mint) NFTs. The user who calls this method becomes the owner of the NFT.

Fungible Tokens:

```
@Validator(<Token Class>)
public async create<Token Name>Token(tokenAsset: <Token Class>) {
  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.save",
"TOKEN");
  return await this.Ctx.ERC1155Token.save(tokenAsset);
}
```

Non-Fungible Tokens:

```
@Validator(<Token Class>, yup.number())
public async create<Token Name>Token(tokenAsset: <Token Class>,
quantity: number) {
  return await this.Ctx.ERC1155Token.save(tokenAsset, quantity);
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.
- `quantity: number` – For non-fungible tokens only, the number of tokens to mint. The only supported value for this parameter is `1`.

Returns:

- On success, the token asset in JSON format, which includes the following information, depending on the token type.
- `tokenMetadata` – JSON information that describes the token.
- `createdBy` – The account ID of the caller, who is the user minting the token. This property cannot be edited.
- `creationDate` – The time stamp of the minting transaction. This property cannot be edited.
- `isBurned` – This property indicates whether the token is burned. This property cannot be edited.
- `tokenName` – The name of the token. This property cannot be edited.
- `tokenDesc` – The description of the token.
- `symbol` – The symbol of the token. This property cannot be edited.
- `tokenStandard` – The standard of the token. This property cannot be edited.
- `tokenType` – The type of the token (fungible or non-fungible). This property cannot be edited.
- `tokenUnit` – The unit of the token (whole or fractional). This property cannot be edited.

- `behaviors` – A list of token behaviors. This property cannot be edited.
- `mintable` – The properties related to minting. The `max_mint_quantity` value defines the maximum number of tokens that can be created for the token class.
- `owner` – The account ID of the current owner, who is the caller of the method.
- `tokenUri` – The URI of the token.
- `quantity` – The quantity of the token.

Return Value Example (Whole NFT):


```
{
    "tokenMetadata": {
        "paintingName": "monalisa",
        "description": "monalisa painting",
        "image": "image link",
        "painterName": "Leonardo da Vinci"
    },
    "assetType": "otoken",
    "quantity": 1,
    "tokenId": "artnft",
    "tokenName": "artcollection",
    "tokenDesc": "artcollection nft",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 500
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2022-12-29T04:08:35.000Z",
    "isBurned": false,
    "tokenUri": "tu",
    "price": 10000,
    "onSaleFlag": false
}
```

Return Value Example (Fungible Token):

```
{
    "assetType": "otoken",
    "tokenId": "Loyalty",
    "tokenName": "loyalty",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 10000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "Dollar"
}
```

Return Value Example (Fractional NFT):

```
{
    "tokenMetadata": {
        "painting_name": "paint",
        "description": "Painting Description"
    },
    "assetType": "otoken",
    "tokenId": "realEstate",
    "tokenName": "realestate",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
```

```
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "quantity": 100,
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2023-06-14T04:20:14.000Z",
    "divisible": {
        "decimal": 2
    },
    "isBurned": false,
    "tokenUri": "www.realestate.example.com",
    "price": 1000,
    "on_sale_flag": true
}
```

**update<Token Name>Token**

This method updates tokens. Every token that is defined has its own update method. You cannot update token metadata or the token URI of non-fungible tokens. For fungible tokens, this method can be called only by a `Token Admin` of the chaincode. For non-fungible tokens, this method can be called only by the token owner.

Fungible Tokens:

```
@Validator(<Token Class>)
public async update<Token Name>Token(tokenAsset: <Token Class>) {
  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.update",
"TOKEN");
  return await this.Ctx.ERC1155Token.update(tokenAsset);
}
```

Non-Fungible Tokens:

```
@Validator(<Token Class>)
public async update<Token Name>Token(tokenAsset: <Token Class>) {
  return await this.Ctx.ERC1155Token.update(tokenAsset);
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.

Returns:

- On success, the updated token asset in JSON format.

Return Value Example (Whole NFT):


```
{
    "tokenMetadata": {
        "paintingName": "monalisa",
        "description": "monalisa painting",
        "image": "image link",
        "painterName": "Leonardo da Vinci"
    },
    "assetType": "otoken",
    "quantity": 1,
    "tokenId": "artnft",
    "tokenName": "artcollection",
    "tokenDesc": "artcollection nft",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 500
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-12-29T04:08:35.000Z",
    "isBurned": false,
    "tokenUri": "tu",
    "price": 10000,
    "onSaleFlag": false
}
```


**getTokenHistory**

This method returns the history for a specified token ID. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async getTokenHistory(tokenId: string) {
```

```
  return await this.Ctx.ERC1155Token.getTokenHistory(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON array that contains the token history.

Return Value Example (Fungible Token):


```
[
    {
        "trxId":
"ef4af760c3d7ee5e273196231d59fb91cafe6ca0f78c64747e87bc9bcbb3334b",
        "timeStamp": "2023-09-04T02:36:20.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "LoyaltyToken",
            "tokenName": "loyalty",
            "tokenDesc": "Updated Fungible Whole",
            "tokenStandard": "erc1155+",
            "tokenType": "fungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 10000
            },
            "divisible": {
                "decimal": 2
            },
            "currency_name": "Rupees"
        }
    },
    {
        "trxId":
"4fb391a8903633a12a545cd2ecfb57f5575241325abf59995e2a4ed96572bb09",
        "timeStamp": "2023-09-04T02:35:07.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "LoyaltyToken",
            "tokenName": "loyalty",
```

```
            "tokenDesc": "Fungible Whole",
            "tokenStandard": "erc1155+",
            "tokenType": "fungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 10000
            },
            "divisible": {
                "decimal": 2
            },
            "currency_name": "Dollar"
        }
    }
]
```


Return Value Example (Fractional NFT):


```
[
    {
        "txId":
"99bca74f401465206da7499cbf704dd443b3c3d94e348b1d6682ab5ee1864a08",
        "timestamp": "2023-06-20T01:09:18.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "FNFT",
            "tokenName": "realestate",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
```

```
                "max_mint_quantity": 20000
            },
            "quantity": 100,
            "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "creationDate": "2023-06-20T00:53:13.000Z",
            "divisible": {
                "decimal": 2
            },
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 2000,
            "on_sale_flag": true,
            "owners": [
                {
                    "accountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
                    "tokenShare": 90
                },
                {
                    "accountId":
"oaccount~3cddfdaa855900579d963aa6f755a4aed1f3a474a2462c1b45bd7f36df673224",
                    "tokenShare": 10
                }
            ]
        }
    },
    {
        "txId":
"d517c61f40e7d6af2f04fe6d337b3e5108eb57030c9dc823793498fd4fed671b",
        "timestamp": "2023-06-20T00:53:13.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "FNFT",
            "tokenName": "realestate",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 100,
```

```
            "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "creationDate": "2023-06-20T00:53:13.000Z",
            "divisible": {
                "decimal": 2
            },
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 2000,
            "on_sale_flag": true,
            "owners": [
                {
                    "accountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
                    "tokenShare": 100
                }
            ]
        }
    }
]
```


Return Value Example (Whole NFT):


```
[
    {
        "trxId":
"92ac6b56112acdba724dd49924d2420a7899c013c61aa40d272e8ab391a65e0f",
        "timeStamp": "2023-09-04T02:28:48.000Z",
        "value": {
            "tokenMetadata": {
                "painting_name": "monalisa",
                "description": "monalisa painting",
                "image": "image link",
                "painter_name": "Leonardo da Vinci"
            },
            "assetType": "otoken",
            "tokenId": "artnft",
            "tokenName": "artcollection",
            "tokenDesc": "Updated Token Description",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
```

```
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 1,
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "creationDate": "2023-09-04T02:27:19.000Z",
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 10000,
            "on_sale_flag": true
        }
    },
    {
        "trxId":
"27697dd4a8dba53bad073aa95587cd1ef173b02fd95d771a60273d301fd3bcbe",
        "timeStamp": "2023-09-04T02:27:19.000Z",
        "value": {
            "tokenMetadata": {
                "painting_name": "monalisa",
                "description": "monalisa painting",
                "image": "image link",
                "painter_name": "Leonardo da Vinci"
            },
            "assetType": "otoken",
            "tokenId": "artnft",
            "tokenName": "artcollection",
            "tokenDesc": "artcollection nft",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 1,
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
```

```
            "creationDate": "2023-09-04T02:27:19.000Z",
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 10000,
            "on_sale_flag": true
        }
    }
]


[
    {
        "trxId":
"ebda1f31543f8906b7ec50a631afff6b8318a3d63c84f3e73be6785cc2ff31ff",
        "timeStamp": "2023-06-20T01:14:08.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "NFT",
            "tokenName": "artcollection",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 1,
            "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "creationDate": "2023-06-20T01:14:08.000Z",
            "owner":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "isBurned": false,
            "tokenUri": "www.NFT.example.com",
            "price": 2000,
            "on_sale_flag": true
        }
```

```
    }
]
```

**`getAllTokens`**

This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
@GetMethod()
@Validator(yup.string())
public async getTokenHistory(tokenId: string) {
  return await this.Ctx.ERC1155Token.getTokenHistory(tokenId);
}
```

Parameters:

- none

Returns:

- A list of all token assets in JSON format.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
```

```
  {
    "key": "tokenTwo",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenTwo",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "art",
    "valueJson": {
      "assetType": "otoken",
      "quantity": 1,
      "tokenId": "art",
      "tokenName": "artcollection",
      "tokenStandard": "erc1155+",
      "tokenType": "nonfungible",
      "tokenUnit": "whole",
      "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter"
      },
      "mintable": {
        "max_mint_quantity": 20000
      },
      "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
```

```
2c6",
      "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
      "creationDate": "2022-12-08T08:52:57.000Z",
      "isBurned": true,
      "tokenUri": "art.example.com",
      "transferredBy":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
      "transferredDate": "2022-12-08T08:59:17.000Z",
      "burnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
      "burnedDate": "2022-12-08T09:01:28.000Z"
    }
  },
  {
    "key": "FNFT",
    "valueJson": {
        "assetType": "otoken",
        "tokenId": "FNFT",
        "tokenName": "realestate",
        "tokenStandard": "erc1155+",
        "tokenType": "nonfungible",
        "tokenUnit": "fractional",
        "behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "roles": {
            "minter_role_name": "minter",
            "burner_role_name": "burner"
        },
        "mintable": {
            "max_mint_quantity": 20000
        },
        "quantity": 100,
        "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
        "creationDate": "2023-06-20T00:53:13.000Z",
        "divisible": {
            "decimal": 2
        },
        "isBurned": false,
        "tokenUri": "www.FNFT.example.com",
        "price": 2000,
        "on_sale_flag": true
    }
  }
]
```

**getTokenById**

This method returns a token object if the token is present in the state database. For fractional NFTs, the list of owners is also returned. This method can be called only by a `Token Admin` of the chaincode or the token owner.

```
@GetMethod()
@Validator(yup.string())
public async getTokenById(tokenId: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getTokenById",
"TOKEN", { tokenId });
  return await this.Ctx.ERC1155Token.getTokenById(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Return Value Example (Whole NFT):

```
{
  "assetType": "otoken",
  "quantity": 1,
  "tokenId": "art",
  "tokenName": "artcollection",
  "tokenStandard": "erc1155+",
  "tokenType": "nonfungible",
  "tokenUnit": "whole",
  "behaviors": [
    "indivisible",
    "singleton",
    "mintable",
    "transferable",
    "burnable",
    "roles"
  ],
  "roles": {
    "minter_role_name": "minter"
  },
  "mintable": {
    "max_mint_quantity": 20000
  },
  "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
  "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
  "creationDate": "2022-12-08T08:52:57.000Z",
  "isBurned": true,
  "tokenUri": "art.example.com",
  "transferredBy":
```

```
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "transferredDate": "2022-12-08T08:59:17.000Z",
  "burnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "burnedDate": "2022-12-08T09:01:28.000Z"
}
```


Return Value Example (Fungible Token):


```
{
    "assetType": "otoken",
    "tokenId": "Loyalty",
    "tokenName": "loyalty",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 10000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "Dollar"
}
```


Return Value Example (Fractional NFT):


```
{
    "tokenMetadata": {
        "painting_name": "paint",
        "description": "Painting Description"
    },
    "assetType": "otoken",
    "tokenId": "realEstate",
    "tokenName": "realestate",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "fractional",
    "behaviors": [
```

```
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "roles": {
            "minter_role_name": "minter",
            "burner_role_name": "burner"
        },
        "mintable": {
            "max_mint_quantity": 20000
        },
        "quantity": 100,
        "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "creationDate": "2023-06-14T04:20:14.000Z",
        "divisible": {
            "decimal": 2
        },
        "isBurned": false,
        "tokenUri": "www.realestate.example.com",
        "price": 1000,
        "on_sale_flag": true,
        "owners": [
            {
                "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
                "tokenShare": 100
            }
        ]
}
```

**getAllTokensByUser**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAllTokensByUser(orgId: string, userId: string) {
  const accountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getAllTokensByUse
r", "TOKEN", { accountId });
  return await this.Ctx.ERC1155Token.getAllTokensByUser(accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "nftToken",
    "valueJson": {
      "assetType": "otoken",
      "quantity": 1,
      "tokenId": "nftToken",
      "tokenName": "artcollection",
      "tokenStandard": "erc1155+",
      "tokenType": "nonfungible",
      "tokenUnit": "whole",
      "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
```

```
      "roles": {
        "minter_role_name": "minter"
      },
      "mintable": {
        "max_mint_quantity": 20000
      },
      "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "creationDate": "2022-12-08T09:10:21.000Z",
      "isBurned": false,
      "tokenUri": "example.com"
    }
  }
]
```

**ownerOf**

This method returns the account ID, organization ID, and user ID of the owner of the specified token ID. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async ownerOf(tokenId: string) {
  return await this.Ctx.ERC1155Token.ownerOf(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example (Whole NFT):

```
{
    "accountId":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
    "orgId": "appdev",
    "userId": "idcqa"
}
```

Return Value Example (Fractional NFT):

```
[
    {
        "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
        "orgId": "Org1MSP",
```

```
        "userId": "admin"
    },
    {
        "accountId":
"oaccount~74108eca702bab6d8548e740254f2cc7955d886885251d52d065042172a59db0",
        "orgId": "Org1MSP",
        "userId": "user"
    }
]
```

**URI**

This method returns the URI of a specified token. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async URI(tokenId: string) {
  return await this.Ctx.ERC1155Token.tokenURI(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "tokenUri": "example.com"
}
```

**name**

This method returns the name of the token class. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async name(tokenId: string) {
  return await this.Ctx.ERC1155Token.name(tokenId);
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{"tokenName": "artcollection"}
```

**totalSupply**
This method returns the total number of minted tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.object())
public async totalSupply(tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.totalSupply",
"TOKEN");
  const token = await
this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.totalSupply(token);
}
```

Parameters:

- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{"totalSupply": 110}
```

**totalNetSupply**
This method returns the total number of minted tokens minus the number of burned tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.object())
public async totalNetSupply(tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.totalNetSupply",
"TOKEN");
  const token = await
this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.totalNetSupply(token);
}
```

Parameters:

- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{"totalNetSupply": 105}
```

**`getTokensByName`**

This method returns all of the token assets for a specified token name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getTokensByName(tokenName: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getTokensByName",
"TOKEN");
  return await this.Ctx.ERC1155Token.getTokensByName(tokenName);
}
```

Parameters:

- `tokenName: string` – The name of the token.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
```

```
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenTwo",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  }
]
```

**getTokenDecimal**

This method returns the number of decimal places for a specified token. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async getTokenDecimal(tokenId: string) {
  const token = await this.Ctx.ERC1155Token.get(tokenId);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getDecimals",
"TOKEN");
  return {
```

```
    msg: `Token Id: ${tokenId} has $
{this.Ctx.ERC1155Token.getDecimals(token)} decimal places.`,
  };
}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "msg": "Token Id: tokenOne has 2 decimal places."
}
```

### Methods for Account Management

**`createAccount`**

This method creates an account for a specified user and associated token accounts for fungible or non-fungible tokens. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user holds. Users must have accounts in the network to complete token-related operations. This method can be called only by a `Token Admin` of the chaincode.
A user account has a unique ID, which is formed by an SHA-256 hash of the `orgId` parameter and the `userId` parameter.
A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).
A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this account.
User account IDs start with with `ouaccount~`. Token account IDs start with `oaccount~`.

```
@Validator(yup.string(), yup.string(), yup.boolean(), yup.boolean())
public async createAccount(orgId: string, userId: string, ftAccount:
boolean, nftAccount: boolean) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.createAccount",
"TOKEN");
  return await this.Ctx.ERC1155Account.createAccount(orgId, userId,
ftAccount, nftAccount);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

- `ftAccount: boolean` – If true, a fungible token account is created and associated with the user account.
- `nftAccount: boolean` – If true, a non-fungible token account is created and associated with the user account.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~cf20877546f52687f387e7c91d88b9722c97e1a456cc0484f40c747f7804
feae",
  "userId": "user1",
  "orgId": "appdev",
  "totalAccounts": 2,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b",
      "tokenId": ""
    }
  ],
  "associatedNftAccount":
"oaccount~73c3e835dac6d0a56ca9d8def08269f83cefd59b9d297fe2cdc5a9083828f
a58"
}
```

**`createUserAccount`**

This method creates an account for a specified user. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations.

An account ID is an SHA-256 hash of the `orgId` parameter and the `userId` parameter. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string())
  public async createUserAccount(orgId: string, userId: string) {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.createUserAccou
nt", "TOKEN");
    return await this.Ctx.ERC1155Account.createUserAccount(orgId,
userId);
  }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object of the user account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 0,
  "totalFtAccounts": 0,
  "associatedFtAccounts": [],
  "associatedNftAccount": ""
}
```

**`createTokenAccount`**

This method creates a fungible or non-fungible token account to associate with a user account.

A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).

A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this account.

This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async createTokenAccount(orgId: string, userId: string, tokenType:
TokenType) {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.createTokenAccount",
"TOKEN");
    return await this.Ctx.ERC1155Account.createTokenAccount(orgId, userId,
tokenType);
  }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

- `tokenType: TokenType` – The type of token account to create. The only supported token types are `nonfungible` and `fungible`.

Returns:

- On success, a JSON object of the token account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
      "tokenId": ""
    }
  ],
  "associatedNftAccount": ""
}
```

**`associateFungibleTokenAccount`**
This method associates a user's fungible token account to a particular fungible token. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async associateFungibleTokenToAccount(orgId: string, userId:
string, tokenId: string) {
  const accountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.associateFungib
leTokenToAccount", "TOKEN", { accountId });
  return await
this.Ctx.ERC1155Account.associateTokenToToken(accountId, tokenId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object of the user account, which shows that the fungible token was associated to the token account. For example, in the following example, the first object in the `associatedFtAccounts` array shows that the fungible token account ID and the token ID are associated.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "tokenId": "tokenOne"
    }
  ],
  "associatedNftAccount": ""
}
```

**`getAccountHistory`**

This method returns history for a specified token account. This is an asynchronous method. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string())
public async getAccountHistory(orgId: string, userId: string, tokenId?:
string) {
  const userAccountId = await
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
  const userAccount = await
this.Ctx.ERC1155Account.getAccount(userAccountId, tokenId);
  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.history",
"TOKEN", { accountId: userAccountId });
  return await
this.Ctx.ERC1155Account.getAccountHistory(userAccount.accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

- `tokenId?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, an array of JSON objects that describes the account history.

Return Value Example:


```
[
  {
    "trxId":
"89f462697f3c988024b2c248cbda21f9eb7e96567e56dd8db64ada96a4845a7f",
    "timeStamp": "2022-12-08T07:15:10.000Z",
    "value": {
      "assetType": "oaccount",
      "bapAccountVersion": 1,
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
      "userId": "user2",
      "orgId": "appdev",
      "tokenType": "fungible",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "balance": 110
    }
  },
  {
    "trxId":
"30dd4fe0746350c85a5000996974487010a0a8fee73d6b2e480c3ca330a6d31f",
    "timeStamp": "2022-12-08T06:43:10.000Z",
    "value": {
      "assetType": "oaccount",
      "bapAccountVersion": 0,
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
      "userId": "user2",
      "orgId": "appdev",
      "tokenType": "fungible",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "balance": 0
    }
  },
  {
    "trxId":
"6226c0455cc3a4f99c3fd7ed8b1d36b8e93f863e42ab61a9b0d399f2d69d2f3d",
    "timeStamp": "2022-12-08T06:41:51.000Z",
    "value": {
      "assetType": "oaccount",
      "bapAccountVersion": 0,
      "accountId":
```

```
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "userId": "user2",
      "orgId": "appdev",
      "tokenType": "fungible",
      "tokenId": "",
      "balance": 0
    }
  }
]
```

**getAccount**

This method returns token account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string())
public async getAccount(orgId: string, userId: string, tokenId?: string) {
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  await this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.getAccount",
"TOKEN", { accountId: userAccountId });
  return await this.Ctx.ERC1155Account.getAccountWithStatus(userAccountId,
tokenId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example (Non-Fungible Token Account):

```
{
  "assetType": "oaccount",
  "bapAccountVersion": 1,
  "status": "active",
  "accountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
  "userId": "idcqa",
  "orgId": "appdev",
  "tokenType": "nonfungible",
  "noOfNfts": 1
}
```

Return Value Example (Fungible Token Account):

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "accountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
  "userId": "idcqa",
  "orgId": "appdev",
  "tokenType": "fungible",
  "tokenId": "t1",
  "tokenName": "loyalty",
  "balance": 100
}
```

**getAllAccounts**

This method returns details of all user accounts. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async getAllAccounts() {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.getAllAccounts"
, "TOKEN");
  return await this.Ctx.ERC1155Account.getAllAccounts();
}
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
        {
            "assetType": "ouaccount",
            "accountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
            "userId": "idcqa",
            "orgId": "appdev",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId":
```

```
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371"
        },
        {
            "assetType": "ouaccount",
            "accountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
            "userId": "user1_minter",
            "orgId": "appdev",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId":
"oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446"
        },
    ]
```

**getAccountDetailsByUser**

This method returns an account summary for a specified user and details of fungible and non-fungible tokens that are associated with the user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getAccountDetailsByUser(orgId: string, userId: string) {
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.getAccountDetailsByUs
er", "TOKEN", {
    accountId: userAccountId,
  });
  return await this.Ctx.ERC1155Account.getAccountDetailsByUser(orgId,
userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes and account summary for the specified user and details of fungible and non-fungible tokens that are associated with the user. For fractional non-fungible tokens, the `tokenShare` property in the `associatedNFTs` section shows the share that the user owns.

Return Value Example:

```
{
    "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
    "associatedFTAccounts": [
        {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "tokenId": "FT",
            "balance": 50
        }
    ],
    "associatedNFTAccount": {
        "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
        "associatedNFTs": [
            {
                "nftTokenId": "FNFT",
                "tokenShare": 100
            },
            {
                "nftTokenId": "FNFT2",
                "tokenShare": 110
            },
            {
                "nftTokenId": "NFT"
            }
        ]
    }
}
```

**`getUserByAccountId`**
This method returns the user details of a specified account ID. This method can be called by any user.

```
@GetMethod()
@Validator(yup.string())
public async getUserByAccountId(accountId: string) {
  return await this.Ctx.ERC1155Account.getUserByAccountId(accountId);
}
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`orgId` and `userId`).

Return Value Example:

```
{
  "orgId": "appdev"
  "userId": "user2",
}
```

### Methods for Role Management

**`addRole`**

This method adds a role to a specified user and token. This method can be called only by a `Token Admin` of the chaincode. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. The specified user must have a token account that is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.object())
public async addRole(orgId: string, userId: string, role: string,
tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.addRoleMember",
"TOKEN");
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  const token = await this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.addRoleMember(role, userAccountId,
token);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the role to add to the specified user.
- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Returns:

- On success, a message with account details.

Return Value Example:

```
{
  "msg": "Successfully added role 'minter' to Account Id:
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d2
2a (Org-Id: appdev, User-Id: idcqa)"
}
```

**isInRole**

This method returns a Boolean value to indicate if a user has a specified role. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account. The specified user must have a token account that is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string(), yup.object())
public async isInRole(orgId: string, userId: string, role: string,
tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.removeRoleMember"
, "TOKEN");
  const userAccountId =
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
  const token = await
this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.isInRole(role, userAccountId,
token);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the role to search for.
- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(Org-Id: appdev, User-Id: idcqa) has minter role"
}
```

**removeRole**

This method removes a role from a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode. The specified user must have a token account that is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.object())
public async removeRole(orgId: string, userId: string, role: string,
tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.removeRoleMember",
"TOKEN");
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  const token = await this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.removeRoleMember(role, userAccountId,
token);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the role to remove from the specified user.
- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{
  "msg": "Successfully removed role 'minter' from Account Id:
oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b
```

```
(Org-Id: appdev, User-Id: user1)"
}
```

**getAccountsByRole**

This method returns a list of all account IDs for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.object())
public async getAccountsByRole(role: string, tokenDetail: TokenDetail)
{
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ROLE.getAccountsByRole"
, "TOKEN");
  const token = await
this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.getAccountsByRole(role, token);
}
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{
  "accounts": [

"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",

"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b"
  ]
}
```

**getUsersByRole**

This method returns a list of all users for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.object())
public async getUsersByRole(role: string, tokenDetail: TokenDetail) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ROLE.getUsersByRole",
"TOKEN");
  const token = await this.Ctx.ERC1155Token.getTokenByIdOrName(tokenDetail);
  return await this.Ctx.ERC1155Token.getUsersByRole(role, token);
}
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetails: TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"tokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"tokenName":"artCollection"}
  ```

Return Value Example:

```
{
    "users": [
        {
            "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "orgId": "appdev",
            "userId": "user2"
        },
        {
            "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
            "orgId": "appdev",
            "userId": "user1"
        }
    ]
}
```

### Methods for Transaction History Management

**getAccountTransactionHistory**

This method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner. For non-fungible tokens, this method can only be called when connected to the remote Oracle Blockchain Platform network.

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string())
public async getAccountTransactionHistory(orgId: string, userId:
string, tokenId?: string) {
  const userAccountId = await
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.getAccountTrans
actionHistory", "TOKEN", {
    accountId: userAccountId,
  });
  const account = await
this.Ctx.ERC1155Account.getAccount(userAccountId, tokenId);
  return await
this.Ctx.ERC1155Account.getAccountTransactionHistory(account.accountId)
;
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Return Value Example:

```
[
    {
        "transactionId":
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddf
ae140bc~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475
b",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 10,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "transactionType": "DEBIT",
```

```
        "balance": 90
    },
    {
        "transactionId":
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddfae140b
c~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc114",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "transactionType": "DEBIT"
    },
    {
        "transactionId":
"otransaction~c369929e28e78de06c72d020f1418c9a154a7dd280b2e22ebb4ea4485e24912
4~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "timestamp": "2023-06-06T14:47:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~114a1bc78d04be48ee6dc140c32c042ee9481cb118959626f090eec74452242
2~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "timestamp": "2023-06-05T17:17:57.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 100,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT",
        "balance": 100
    }
]
```


**getTransactionById**

This method returns the transaction details for a specified transaction ID. Anyone can call this method.

```
@GetMethod()
@Validator(yup.string())
public async getTransactionById(transactionId: string) {
  return await this.Ctx.ERC1155Transaction.getTransactionById(transactionId);
}
```

Parameters:

- `transactionId: string` – The ID of the transaction.

Return Value Example:

```
{
  "transactionId":
"otransaction~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe8866
1fe3cbe~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a5
3",
  "history": [
    {
      "trxId":
"9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cbe",
      "timeStamp": "2022-12-08T09:01:28.000Z",
      "value": {
        "assetType": "otransaction",
        "transactionId":
"otransaction~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe8866
1fe3cbe~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a5
3",
        "tokenId": "tokenOne",
        "fromAccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
        "toAccountId": "",
        "transactionType": "BURN",
        "amount": 5,
        "timestamp": "2022-12-08T09:01:28.000Z",
        "triggeredByUserAccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc"
      }
    }
  ]
}
```

**`deleteHistoricalTransactions`**

This method deletes transactions before a specified time stamp from the state database. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.date())
public async deleteHistoricalTransactions(time_to_expiration: Date) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("TRANSACTION.deleteTransactions
", "TOKEN");
  return await
this.Ctx.ERC1155Transaction.deleteTransactions(time_to_expiration);
}
```

Parameters:

- `timestamp: string` – All transactions before this time stamp will be deleted.

Return Value Example:

```
{
  "msg": "Successfuly deleted transaction older than date: Thu Apr 07 2022
21:18:59 GMT+0000 (Coordinated Universal Time).",
  "transactions": [

"otransaction~30513757d8b647fffaafac440d743635f5c1b2e41b25ebd6b70b5bbf78a2643
f",

"otransaction~ac0e908c735297941ba58bb208ee61ff4816a1e54c090d68024f82adf743892
b"
  ]
}
```

### Methods for Token Behavior Management - Mintable Behavior

**`mintBatch`**
This method creates (mints) multiple tokens in a batch operation. This method creates only fungible tokens or fractional non-fungible tokens.
For fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. You cannot mint more than the `max_mint_quantity` property of the token, if that property was specified when the token was created or updated.
For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. Additionally, the caller must also be the creator of the token. There is no upper limit to the quantity of fractional non-fungible tokens that can be minted.
You cannot use this method to mint a whole non-fungible token.

```
@Validator(yup.string(), yup.string(), yup.array().of(yup.string()),
yup.array().of(yup.number()))
public async mintBatch(orgId: string, userId: string, tokenIds: string[],
quantity: number[]) {
  const accountId = this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
  return await this.Ctx.ERC1155Token.mintBatch(accountId, tokenIds,
quantity);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenIds: string[]` – The list of token IDs to mint tokens for.
- `quantity: number[]` – The list of quantities of tokens to mint, corresponding to the token ID array.

Returns:

- On success, a JSON object that includes details on the minted tokens.

Return Value Example:

```
{
    "msg": "Successfully minted batch of tokens for User-Account-Id
ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51
f38 (Org-Id: appdev, User-Id: idcqa).",
    "details": [
        {
            "msg": "Successfully minted 100 tokens of fractional
tokenId: plot55 to Org-Id: appdev, User-Id: idcqa"
        },
        {
            "msg": "Successfully minted 100 tokens of tokenId: loyalty
to Token-Account-Id
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d41
0e"
        }
    ]
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`batchTransferFrom`**
This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
This method does not validate that the caller of the method is the specified sender. This method can be called by any user.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.array().of(yup.string()), yup.array().of(yup.number()))
public async batchTransferFrom(
  fromOrgId: string,
  fromUserId: string,
  toOrgId: string,
  toUserId: string,
  tokenIds: string[],
  quantity: number[]
) {
  const fromAccountId =
this.Ctx.ERC1155Account.generateAccountId(fromOrgId, fromUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
  const toAccountId =
this.Ctx.ERC1155Account.generateAccountId(toOrgId, toUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
```

```
  return await this.Ctx.ERC1155Token.batchTransferFrom(fromAccountId,
toAccountId, tokenIds, quantity);
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender and token owner in the current organization.
- `fromUserId: string` – The user name or email ID of the sender and token owner.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenIds: string[]` – A list of token IDs for the tokens to transfer.
- `quantity: number[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**`safeBatchTransferFrom`**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.

For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
The caller of the method must be the specified sender. This method can be called by any user.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.array().of(yup.string()), yup.array().of(yup.number()))
public async safeBatchTransferFrom(
  fromOrgId: string,
  fromUserId: string,
  toOrgId: string,
  toUserId: string,
  tokenIds: string[],
  quantity: number[]
) {
  const fromAccountId =
this.Ctx.ERC1155Account.generateAccountId(fromOrgId, fromUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
  const toAccountId =
this.Ctx.ERC1155Account.generateAccountId(toOrgId, toUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
  return await
this.Ctx.ERC1155Token.safeBatchTransferFrom(fromAccountId,
toAccountId, tokenIds, quantity);
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender and token owner in the current organization.
- `fromUserId: string` – The user name or email ID of the sender and token owner.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `tokenIds: string[]` – A list of token IDs for the tokens to transfer.
- `quantity: number[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
```

```
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**balanceOfBatch**

This method completes a batch operation that gets the balance of token accounts. The account details are specified in three separate lists of organization IDs, user IDs, and token IDs. This method can be called only by a `Token Admin` of the chaincode or by account owners. Account owners can see balance details only for accounts that they own.

```
@GetMethod()
@Validator(yup.array().of(yup.string()), yup.array().of(yup.string()),
yup.array().of(yup.string()))
public async balanceOfBatch(orgIds: string[], userIds: string[], tokenIds:
string[]) {
  let callerAccountCheck = false;
  try {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.balanceOfBatch",
"TOKEN");
  } catch (err) {
    callerAccountCheck = true;
  }
  const accountIds = await
this.Ctx.ERC1155Account.generateAccountIds(orgIds, userIds,
callerAccountCheck);
  return await this.Ctx.ERC1155Account.balanceOfBatch(accountIds, tokenIds);
}
```

Parameters:

- `orgIds: string[]` – A list of the membership service provider (MSP) IDs in the current organization.

- `userIds: string[]` – A list of the user name or email IDs.
- `tokenIds: string[]` – A list of the token IDs.

Return Value Example:
In the following example, the token ID `FNFT` represents a fractional non-fungible token and the token ID `FT` represents a fungible token.


```
[
    {
        "orgId": "appdev",
        "userId": "idcqa",
        "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "tokenAccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
        "tokenId": "FNFT",
        "balance": 100
    },
    {
        "orgId": "appdev",
        "userId": "idcqa",
        "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "tokenAccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
        "tokenId": "FT",
        "balance": 50
    },
    {
        "orgId": "appdev",
        "userId": "user1_minter",
        "userAccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352
003b",
        "tokenAccountId":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "tokenId": "FNFT",
        "balance": 10
    }
]
```


**`exchangeToken`**
This method exchanges tokens between specified accounts. This method only supports exchanging between an NFT and a fungible token or a fungible token and an

NFT. The NFT can be whole or fractional. This method can be called only by the account owner.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.number(),
yup.string(), yup.string(), yup.string(), yup.number())
  public async exchangeToken(
    fromTokenId: string,
    fromOrgId: string,
    fromUserId: string,
    fromTokenQuantity: number,
    toTokenId: string,
    toOrgId: string,
    toUserId: string,
    toTokenQuantity: number
  ) {
    const fromUserAccountId =
this.Ctx.ERC1155Account.generateAccountId(fromOrgId, fromUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT.exchangeToken",
"TOKEN", { accountId: fromUserAccountId });
    const toUserAccountId =
this.Ctx.ERC1155Account.generateAccountId(toOrgId, toUserId,
ACCOUNT_TYPE.USER_ACCOUNT);
    return await this.Ctx.ERC1155Token.exchangeToken(
      fromTokenId,
      fromUserAccountId,
      fromTokenQuantity,
      toTokenId,
      toUserAccountId,
      toTokenQuantity
    );
  }
```

Parameters:

- `fromTokenId: string` – The ID of the token that the sender owns.
- `fromOrgId: string` – The membership service provider (MSP) ID of the sender in the current organization.
- `fromUserId: string` – The user name or email ID of the sender.
- `fromTokenQuantity: number` – The quantity of tokens from the sender to exchange with the receiver.
- `toTokenId: string` – The ID of the token that the receiver owns.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `toTokenQuantity: number` – The quantity of tokens from the receiver to exchange with the sender.

Returns:

- On success, a message with token exchange details.

Return Value Example:

```
{
    "msg": "Succesfully exchanged 10 tokens of type nonfungible with
tokenId: [r1] from Account
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a100973
71 (OrgId: appdev, UserId: idcqa) to 10 tokens of type fungible with
tokenId: [loy1] from Account
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f62823797
4c (OrgId: appdev, UserId: user1_minter)"
}
```

### Methods for Token Behavior Management - Burnable Behavior

**`burnBatch`**
This method deactivates, or burns, the specified fungible and non-fungible tokens. Any user with the burner role can call this method.

```
@Validator(yup.string(), yup.string(), yup.array().of(yup.string()),
yup.array().of(yup.number()))
public async burnBatch(orgId: string, userId: string, tokenIds:
string[], quantity: number[]) {
    const accountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
    return await this.Ctx.ERC1155Token.burn(accountId, tokenIds,
quantity);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID in the current organization.
- `userId: string` – The user name or email ID.
- `tokenIds: string[]` – The list of the token IDs to burn
- `quantity: number[]` – The list of quantities of tokens to burn, corresponding to the token ID array..

Returns:

- On success, a message with details about the burn operations.

Return Value Example:

```
[
  {
    "msg": "Successfully burned NFT token: 'art' from Account-Id:
oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282
c6 (Org-Id: appdev, User-Id: idcqa)"
  },
```

```
  {
    "msg": "Successfully burned 5 tokens of tokenId: tokenOne from Account-
ID oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(Org-Id: appdev, User-Id: idcqa)"
  },
  {
    "msg": "Successfully burned 2 token share of tokenId: FNFT from Account-
ID oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a
(Org-Id: AutoF1377358917, User-Id: idcqa)"
  }
]
```

**burnNFT**

This method deactivates, or burns, the specified non-fungible token, and returns a token object and token history. Any user with the burner role can call this method.

```
@Validator(yup.string(), yup.string(), yup.string())
public async burnNFT(orgId: string, userId: string, tokenId: string):
Promise<any> {
    const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId);
    const token = await this.Ctx.ERC1155Token.get(tokenId);
    if (token.tokenType !== TOKEN_TYPE.NON_FUNGIBLE) {

      throw new Error(`The Token with id ${tokenId} is not a nonfungible
token`);
    }
    if (token.isBurned === true) {
      throw new Error(`token with tokenId ${tokenId} is already burned`);
    }
    let tokenBurnQuantity = 1;
    const tokenUnit = token.tokenUnit;
    if (tokenUnit === TOKEN_UNIT.FRACTIONAL) {
       const owners = await
this.Ctx.ERC1155Token.getFractionalNFTOwners(tokenId);
       if (owners.length !== 1) {
         throw new Error(`Token with tokenId ${tokenId} has multiple
owners`);
       }
      tokenBurnQuantity = token.quantity;
    }
    const tokenHistory = await
this.Ctx.ERC1155Token.getTokenHistory(tokenId);
    await this.Ctx.ERC1155Token.burn(userAccountId, [tokenId],
[tokenBurnQuantity]);
    token.tokenId = parseInt(token.tokenId);
    if(Number.isNaN(token.tokenId)) {
      throw new Error(`tokenId is expected to be integer but found $
{tokenId}`)
    }
    token.isBurned = true;
```

```
    return {...token, tokenHistory: JSON.stringify(tokenHistory)};
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID in the current organization.
- `userId: string` – The user name or email ID.
- `tokenId: string` – The ID of the non-fungible token to burn

Returns:

- On success, a token object in JSON format that includes token history information.

Return Value Example:

```
{
    "assetType": "otoken",
    "tokenId": 1,
    "tokenName": "artcollection",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "quantity": 1,
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2023-08-22T07:32:40.000Z",
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "isBurned": true,
    "tokenUri": "example.com",
    "price": 120,
    "onSaleFlag": false,
    "tokenHistory":
"[{\"trxId\":\"21a932750f2d4ccffd62eda5678a577cadde0513ed7c7a307f24cd77
```

```
13a1818b\",\"timeStamp\":\"2023-08-22T07:32:40.000Z\",\"value\":
{\"assetType\":\"otoken\",\"tokenId\":\"1\",\"tokenName\":\"artcollection\",\
"tokenStandard\":\"erc1155+
\",\"tokenType\":\"nonfungible\",\"tokenUnit\":\"whole\",\"behaviors\":
[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"burnable\",\"r
oles\"],\"roles\":{\"minter_role_name\":\"minter\"},\"mintable\":
{\"max_mint_quantity\":20000},\"quantity\":1,\"createdBy\":\"oaccount~42e89f4
c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d\",\"creationDate\":
\"2023-08-22T07:32:40.000Z\",\"owner\":\"oaccount~42e89f4c72dfde9502814876423
c6da630d466e87436dd1aae201d347ad1288d\",\"isBurned\":false,\"tokenUri\":\"exa
mple.com\",\"price\":120,\"onSaleFlag\":false}}]"
}
```


### SDK Methods

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

**`checkAuthorization`**
Use this method to add an access control check to an operation. This is an asynchronous function. Certain token methods can be run only by the `Token Admin` or `AccountOwner` of the token or by the `MultipleAccountOwner` for users with multiple accounts. The access control mapping is described in the `../lib/constant.ts` file. You can modify access control by editing the `../lib/constant.ts` file. To use your own access control or to disable access control, remove the access control code from the automatically generated controller methods and custom methods.

```
ADMIN: {
   isUserTokenAdmin: ["Admin"],
   addAdmin: ["Admin"],
   removeAdmin: ["Admin"],
   getAllAdmins: ["Admin"],
 },
 TOKEN: {
   save: ["Admin"],
   getAllTokens: ["Admin"],
   get: ["Admin"],
   update: ["Admin"],
   getDecimals: ["Admin"],
   getTokensByName: ["Admin"],
```

```
  addRoleMember: ["Admin"],
  removeRoleMember: ["Admin"],
  isInRole: ["Admin", "AccountOwner"],
  getTotalMintedTokens: ["Admin"],
  getNetTokens: ["Admin"],
  getTokenHistory: ["Admin"],
},
ROLE: {
  getAccountsByRole: ["Admin"],
  getUsersByRole: ["Admin"],
},
TRANSACTION: {
  deleteTransactions: ["Admin"],
},
ACCOUNT: {
  createAccount: ["Admin"],
  associateToken: ["Admin"],
  getAllAccounts: ["Admin"],
  getAccountsByUser: ["Admin", "MultipleAccountOwner"],
  getAccount: ["Admin", "AccountOwner"],
  history: ["Admin", "AccountOwner"],
  getAccountTransactionHistory: ["Admin", "AccountOwner"],
  getAccountTransactionHistoryWithFilters: ["Admin", "AccountOwner"],
  getSubTransactionsById: ["Admin", TRANSACTION_INVOKER],
  getSubTransactionsByIdWithFilters: ["Admin", TRANSACTION_INVOKER],
  getAccountBalance: ["Admin", "AccountOwner"],
  getAccountOnHoldBalance: ["Admin", "AccountOwner"],
  getOnHoldIds: ["Admin", "AccountOwner"],
  getConversionHistory: ["Admin", "AccountOwner"],
},
ACCOUNT_STATUS: {
  get: ["Admin", "AccountOwner"],
  history: ["Admin", "AccountOwner"],
  activateAccount: ["Admin"],
  suspendAccount: ["Admin"],
  deleteAccount: ["Admin"],
},
TOKEN_CONVERSION: {
  initializeExchangePoolUser: ["Admin"],
  addConversionRate: ["Admin"],
  updateConversionRate: ["Admin"],
  getConversionRate: ["Admin", "AnyAccountOwner"],
  getConversionRateHistory: ["Admin", "AnyAccountOwner"],
  tokenConversion: ["Admin", "AnyAccountOwner"],
  getExchangePoolUser: ["Admin"],
},
ERC721ADMIN: {
  isUserTokenAdmin: ["Admin"],
  addAdmin: ["Admin"],
  removeAdmin: ["Admin"],
  getAllAdmins: ["Admin"],
},
ERC721TOKEN: {
```

```
  getAllTokens: ["Admin"],
  getAllTokensByUser: ["Admin", "AccountOwner"],
  get: ["Admin", TOKEN_OWNER],
  getTokensByName: ["Admin"],
  addRoleMember: ["Admin"],
  removeRoleMember: ["Admin"],
  isInRole: ["Admin", "AccountOwner"],
  totalSupply: ["Admin"],
  totalNetSupply: ["Admin"],
  history: ["Admin"],
},
ERC721ROLE: {
  getAccountsByRole: ["Admin"],
  getUsersByRole: ["Admin"],
},
ERC721TRANSACTION: {
  deleteTransactions: ["Admin"],
},
ERC721ACCOUNT: {
  createAccount: ["Admin"],
  getAllAccounts: ["Admin"],
  getAccountByUser: ["Admin", "MultipleAccountOwner"],
  history: ["Admin", "AccountOwner"],
  getAccountTransactionHistory: ["Admin", "AccountOwner"],
  getAccountTransactionHistoryWithFilters: ["Admin", "AccountOwner"],
  balanceOf: ["Admin", "MultipleAccountOwner"],
},
ERC1155ADMIN: {
  isUserTokenAdmin: ["Admin"],
  addAdmin: ["Admin"],
  removeAdmin: ["Admin"],
  getAllAdmins: ["Admin"],
},
ERC1155TOKEN: {
  getAllTokens: ["Admin"],
  get: ["Admin", TOKEN_OWNER],
  getAllTokensByUser: ["Admin", "AccountOwner"],
  totalSupply: ["Admin"],
  totalNetSupply: ["Admin"],
  getTokensByName: ["Admin"],
  getDecimals: ["Admin"],
  addRoleMember: ["Admin"],
  removeRoleMember: ["Admin"],
  isInRole: ["Admin", "AccountOwner"],
  save: ["Admin"],
  update: ["Admin"],
},
ERC1155ACCOUNT: {
  createAccount: ["Admin"],
  createUserAccount: ["Admin"],
  createTokenAccount: ["Admin"],
  associateFungibleTokenToAccount: ["Admin", "AccountOwner"],
  getAccountsByUser: ["Admin", "AccountOwner"],
```

```
    getAccount: ["Admin", "AccountOwner"],
    history: ["Admin", "AccountOwner"],
    getAllAccounts: ["Admin"],
    balanceOfBatch: ["Admin"],
    getAccountTransactionHistory: ["Admin", "AccountOwner"],
    getAccountTransactionHistoryWithFilters: ["Admin", "AccountOwner"],
    exchangeToken: ["AccountOwner"],
    getAccountDetailsByUser: ["Admin", "AccountOwner"],
  },
  ERC1155ROLE: {
    getAccountsByRole: ["Admin"],
    getUsersByRole: ["Admin"],
  },
```

```
Ctx.ERC1155Auth.checkAuthorization(classFuncName: string, ...args)
```

Parameters:

- `classFuncName: string` – The map value between the class and methods as described in the `../lib/constant.ts` file.
- `...args` – A variable argument where `args[0]` takes the constant `'TOKEN'` and `args[1]` takes the `account_id` to add an access control check for an `AccountOwner`. To add an access control check for a `MultipleAccountOwner`, `args[1]` takes the `org_id` and `args[2]` takes the `user_id`.

Returns:

- On success, a promise. On error, a rejection with an error message.

**`isUserTokenAdmin`**
This method returns the Boolean value `true` if the specified user is a `Token Admin`, and `false` otherwise. The method can be called only by a `Token Admin` of the token chaincode.

```
Ctx.ERC1155Auth.isUserTokenAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
{
  "result": true
}
```

**addAdmin**

This method adds a user as a `Token Admin` of the token chaincode. The method can be called only by a `Token Admin` of the token chaincode.

```
Ctx.ERC1155Admin.addAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user added as a `Token Admin` of the token chaincode.

Return Value Example:

```
{
  "msg": "Successfully added Admin (OrgId: appDev, UserId: user1)"
}
```

**removeAdmin**

This method removes a user as a `Token Admin` of the token chaincode. The method can be called only by a `Token Admin` of the token chaincode. You cannot remove yourself as a `Token Admin`.

```
Ctx.ERC1155Admin.removeAdmin(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user removed as a `Token Admin` of the token chaincode.

Return Value Example:

```
{
  "msg": "Successfully removed Admin (OrgId: appDev, UserId: user1)"
}
```

**getAllAdmins**
This method returns a list of all `Token Admin` users.

```
Ctx.ERC1155Admin.getAllAdmins()
```

Parameters:

- none

Returns:

- On success, a list of all `Token Admin` users, identifed by organization ID and user ID.

Return Value Example:

```
{
  "admins": [
    {
      "orgId": "appdev",
      "userId": "idcqa"
    },
    {
      "orgId": "appdev",
      "userId": "user1"
    }
  ]
}
```

### Methods for Token Configuration Management

**save**
This method creates tokens. Every token that is defined has its own create method. For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method to create an NFT. If not, any user can use this method to create (mint) NFTs. The user who calls this method becomes the owner of the NFT (whole or fractional).

```
Ctx.ERC1155Token.save(tokenAsset: <Token Class>, quantity?: number);
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.
- `quantity: number` – For non-fungible tokens only, the number of tokens to mint. The only supported value for this parameter is `1`.

Returns:

- On success, the token asset in JSON format, which can include the following information.
- `tokenMetadata` – JSON information that describes the token.

- `createdBy` – The account ID of the caller, who is the user minting the token. This property cannot be edited.
- `creationDate` – The time stamp of the minting transaction. This property cannot be edited.
- `isBurned` – This property indicates whether the token is burned. This property cannot be edited.
- `tokenName` – The name of the token. This property cannot be edited.
- `tokenDesc` – The description of the token.
- `symbol` – The symbol of the token. This property cannot be edited.
- `tokenStandard` – The standard of the token. This property cannot be edited.
- `tokenType` – The type of the token (fungible or non-fungible). This property cannot be edited.
- `tokenUnit` – The unit of the token (whole or fractional). This property cannot be edited.
- `behaviors` – A list of token behaviors. This property cannot be edited.
- `mintable` – The properties related to minting. The `max_mint_quantity` value defines the maximum number of tokens that can be created for the token class.
- `owner` – The account ID of the current owner, who is the caller of the method.
- `tokenUri` – The URI of the token.
- `quantity` – The quantity of the token.

Return Value Example (Whole NFT):

```
{
    "tokenMetadata": {
        "paintingName": "monalisa",
        "description": "monalisa painting",
        "image": "image link",
        "painterName": "Leonardo da Vinci"
    },
    "assetType": "otoken",
    "quantity": 1,
    "tokenId": "artnft",
    "tokenName": "artcollection",
    "tokenDesc": "artcollection nft",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
```

```
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 500
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-12-29T04:08:35.000Z",
    "isBurned": false,
    "tokenUri": "tu",
    "price": 10000,
    "onSaleFlag": false
}
```


Return Value Example (Fungible Token):


```
{
    "assetType": "otoken",
    "tokenId": "Loyalty",
    "tokenName": "loyalty",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 10000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "Dollar"
}
```

Return Value Example (Fractional NFT):

```
{
    "tokenMetadata": {
        "painting_name": "paint",
        "description": "Painting Description"
    },
    "assetType": "otoken",
    "tokenId": "realEstate",
    "tokenName": "realestate",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "quantity": 100,
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2023-06-14T04:20:14.000Z",
    "divisible": {
        "decimal": 2
    },
    "isBurned": false,
    "tokenUri": "www.realestate.example.com",
    "price": 1000,
    "on_sale_flag": true
}
```

**update**

This method updates tokens. You cannot update token metadata or the token URI of non-fungible tokens.

```
Ctx.ERC1155Token.update(tokenAsset: any);
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.

Returns:

- On success, the updated token asset in JSON format.

Return Value Example (Whole NFT):


```
{
    "tokenMetadata": {
        "paintingName": "monalisa",
        "description": "monalisa painting",
        "image": "image link",
        "painterName": "Leonardo da Vinci"
    },
    "assetType": "otoken",
    "quantity": 1,
    "tokenId": "artnft",
    "tokenName": "artcollection",
    "tokenDesc": "artcollection nft",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "whole",
    "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 500
    },
    "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "creationDate": "2022-12-29T04:08:35.000Z",
    "isBurned": false,
    "tokenUri": "tu",
    "price": 10000,
    "onSaleFlag": false
}
```


**history (Token)**

This method returns the history for a specified token ID.

```
Ctx.ERC1155Token.history(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON array that contains the token history.

Return Value Example (Fungible Token):


```
[
    {
        "trxId":
"ef4af760c3d7ee5e273196231d59fb91cafe6ca0f78c64747e87bc9bcbb3334b",
        "timeStamp": "2023-09-04T02:36:20.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "LoyaltyToken",
            "tokenName": "loyalty",
            "tokenDesc": "Updated Fungible Whole",
            "tokenStandard": "erc1155+",
            "tokenType": "fungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 10000
            },
            "divisible": {
                "decimal": 2
            },
            "currency_name": "Rupees"
        }
    },
    {
        "trxId":
"4fb391a8903633a12a545cd2ecfb57f5575241325abf59995e2a4ed96572bb09",
        "timeStamp": "2023-09-04T02:35:07.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "LoyaltyToken",
            "tokenName": "loyalty",
            "tokenDesc": "Fungible Whole",
            "tokenStandard": "erc1155+",
            "tokenType": "fungible",
            "tokenUnit": "fractional",
            "behaviors": [
```

```
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 10000
            },
            "divisible": {
                "decimal": 2
            },
            "currency_name": "Dollar"
        }
    }
]
```

Return Value Example (Fractional NFT):

```
[
    {
        "txId":
"99bca74f401465206da7499cbf704dd443b3c3d94e348b1d6682ab5ee1864a08",
        "timestamp": "2023-06-20T01:09:18.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "FNFT",
            "tokenName": "realestate",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 100,
            "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
```

```
04a",
            "creationDate": "2023-06-20T00:53:13.000Z",
            "divisible": {
                "decimal": 2
            },
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 2000,
            "on_sale_flag": true,
            "owners": [
                {
                    "accountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
                    "tokenShare": 90
                },
                {
                    "accountId":
"oaccount~3cddfdaa855900579d963aa6f755a4aed1f3a474a2462c1b45bd7f36df673224",
                    "tokenShare": 10
                }
            ]
        }
    },
    {
        "txId":
"d517c61f40e7d6af2f04fe6d337b3e5108eb57030c9dc823793498fd4fed671b",
        "timestamp": "2023-06-20T00:53:13.000Z",
        "value": {
            "assetType": "otoken",
            "tokenId": "FNFT",
            "tokenName": "realestate",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 100,
            "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "creationDate": "2023-06-20T00:53:13.000Z",
            "divisible": {
```

```
                "decimal": 2
            },
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 2000,
            "on_sale_flag": true,
            "owners": [
                {
                    "accountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
                    "tokenShare": 100
                }
            ]
        }
    }
]
```


Return Value Example (Whole NFT):


```
[
    {
        "trxId":
"92ac6b56112acdba724dd49924d2420a7899c013c61aa40d272e8ab391a65e0f",
        "timeStamp": "2023-09-04T02:28:48.000Z",
        "value": {
            "tokenMetadata": {
                "painting_name": "monalisa",
                "description": "monalisa painting",
                "image": "image link",
                "painter_name": "Leonardo da Vinci"
            },
            "assetType": "otoken",
            "tokenId": "artnft",
            "tokenName": "artcollection",
            "tokenDesc": "Updated Token Description",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
```

```
            },
            "quantity": 1,
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "creationDate": "2023-09-04T02:27:19.000Z",
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
            "price": 10000,
            "on_sale_flag": true
        }
    },
    {
        "trxId":
"27697dd4a8dba53bad073aa95587cd1ef173b02fd95d771a60273d301fd3bcbe",
        "timeStamp": "2023-09-04T02:27:19.000Z",
        "value": {
            "tokenMetadata": {
                "painting_name": "monalisa",
                "description": "monalisa painting",
                "image": "image link",
                "painter_name": "Leonardo da Vinci"
            },
            "assetType": "otoken",
            "tokenId": "artnft",
            "tokenName": "artcollection",
            "tokenDesc": "artcollection nft",
            "tokenStandard": "erc1155+",
            "tokenType": "nonfungible",
            "tokenUnit": "whole",
            "behaviors": [
                "indivisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner"
            },
            "mintable": {
                "max_mint_quantity": 20000
            },
            "quantity": 1,
            "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "creationDate": "2023-09-04T02:27:19.000Z",
            "owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "isBurned": false,
            "tokenUri": "www.FNFT.example.com",
```

```
            "price": 10000,
            "on_sale_flag": true
        }
    }
]
```

**getAllTokens**

This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.getAllTokens()
```

Parameters:

- none

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "assetType": "otoken",
```

```
      "tokenId": "tokenTwo",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "art",
    "valueJson": {
      "assetType": "otoken",
      "quantity": 1,
      "tokenId": "art",
      "tokenName": "artcollection",
      "tokenStandard": "erc1155+",
      "tokenType": "nonfungible",
      "tokenUnit": "whole",
      "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter"
      },
      "mintable": {
        "max_mint_quantity": 20000
      },
      "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
      "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
      "creationDate": "2022-12-08T08:52:57.000Z",
      "isBurned": true,
```

```
      "tokenUri": "art.example.com",
      "transferredBy":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
      "transferredDate": "2022-12-08T08:59:17.000Z",
      "burnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "burnedDate": "2022-12-08T09:01:28.000Z"
    }
  },
  {
    "key": "FNFT",
    "valueJson": {
        "assetType": "otoken",
        "tokenId": "FNFT",
        "tokenName": "realestate",
        "tokenStandard": "erc1155+",
        "tokenType": "nonfungible",
        "tokenUnit": "fractional",
        "behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "roles": {
            "minter_role_name": "minter",
            "burner_role_name": "burner"
        },
        "mintable": {
            "max_mint_quantity": 20000
        },
        "quantity": 100,
        "createdBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
        "creationDate": "2023-06-20T00:53:13.000Z",
        "divisible": {
            "decimal": 2
        },
        "isBurned": false,
        "tokenUri": "www.FNFT.example.com",
        "price": 2000,
        "on_sale_flag": true
    }
  },
]
```

**`get (Token)`**

This method returns a token object if the token is present in the state database. This method can be called only by a `Token Admin` of the chaincode or the token owner.

```
this.Ctx.ERC1155Token.get(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Return Value Example (Whole NFT):

```
{
  "assetType": "otoken",
  "quantity": 1,
  "tokenId": "art",
  "tokenName": "artcollection",
  "tokenStandard": "erc1155+",
  "tokenType": "nonfungible",
  "tokenUnit": "whole",
  "behaviors": [
    "indivisible",
    "singleton",
    "mintable",
    "transferable",
    "burnable",
    "roles"
  ],
  "roles": {
    "minter_role_name": "minter"
  },
  "mintable": {
    "max_mint_quantity": 20000
  },
  "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "creationDate": "2022-12-08T08:52:57.000Z",
  "isBurned": true,
  "tokenUri": "example.com",
  "transferredBy":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "transferredDate": "2022-12-08T08:59:17.000Z",
  "burnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "burnedDate": "2022-12-08T09:01:28.000Z"
}
```

Return Value Example (Fungible Token):

```
{
    "assetType": "otoken",
    "tokenId": "Loyalty",
    "tokenName": "loyalty",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "fungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 10000
    },
    "divisible": {
        "decimal": 2
    },
    "currency_name": "Dollar"
}
```

Return Value Example (Fractional NFT):

```
{
    "tokenMetadata": {
        "painting_name": "paint",
        "description": "Painting Description"
    },
    "assetType": "otoken",
    "tokenId": "realEstate",
    "tokenName": "realestate",
    "tokenDesc": "Token Description",
    "tokenStandard": "erc1155+",
    "tokenType": "nonfungible",
    "tokenUnit": "fractional",
    "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "roles": {
```

```
        "minter_role_name": "minter",
        "burner_role_name": "burner"
    },
    "mintable": {
        "max_mint_quantity": 20000
    },
    "quantity": 100,
    "createdBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "creationDate": "2023-06-14T04:20:14.000Z",
    "divisible": {
        "decimal": 2
    },
    "isBurned": false,
    "tokenUri": "www.realestate.example.com",
    "price": 1000,
    "on_sale_flag": true,
    "owners": [
        {
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "tokenShare": 100
        }
    ]
}
```

**`getAllTokensByUser`**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.getAllTokensByUser(accountId: string)
```

Parameters:

- `accountId: string` – The account ID of the user.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
```

```
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "nftToken",
    "valueJson": {
      "assetType": "otoken",
      "quantity": 1,
      "tokenId": "nftToken",
      "tokenName": "artcollection",
      "tokenStandard": "erc1155+",
      "tokenType": "nonfungible",
      "tokenUnit": "whole",
      "behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter"
      },
      "mintable": {
        "max_mint_quantity": 20000
      },
      "owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "createdBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "creationDate": "2022-12-08T09:10:21.000Z",
      "isBurned": false,
      "tokenUri": "example.com"
    }
  }
]
```

**ownerOf**
This method returns the account ID, organization ID, and user ID of the owner of the specified token ID.

```
Ctx.ERC1155Token.ownerOf(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "accountId":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
    "orgId": "appdev",
    "userId": "idcqa"
}
```

**tokenURI**
This method returns the URI of a specified token. Anyone can call this method.

```
Ctx.ERC1155Token.tokenURI(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "tokenUri": "example.com"
}
```

**name**
This method returns the name of the token class. Anyone can call this method.

```
Ctx.ERC1155Token.name(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{"tokenName": "artcollection"}
```

**totalSupply**
This method returns the total number of minted tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.totalSupply(token: any)
```

Parameters:

- `token: any` – The token asset.

Return Value Example:

```
{"totalSupply": 110}
```

**totalNetSupply**
This method returns the total number of minted tokens minus the number of burned tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.totalNetSupply(token: any)
```

Parameters:

- `token: any` – The token asset.

Return Value Example:

```
{"totalNetSupply": 105}
```

**getTokensByName**
This method returns all of the token assets for a specified token name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.getTokensByName(tokenName: string)
```

Parameters:

- `tokenName: string` – The name of the token.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenOne",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
```

```
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "assetType": "otoken",
      "tokenId": "tokenTwo",
      "tokenName": "moneytok",
      "tokenStandard": "erc1155+",
      "tokenType": "fungible",
      "tokenUnit": "fractional",
      "behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "roles": {
        "minter_role_name": "minter",
        "burner_role_name": "burner"
      },
      "mintable": {
        "max_mint_quantity": 1000
      },
      "divisible": {
        "decimal": 2
      }
    }
  }
]
```

**`getDecimals`**
This method returns the number of decimal places for a specified token. If the divisible behavior is not specified for the token, then the default value of zero decimal places is returned.

```
Ctx.ERC1155Token.getDecimals(token)
```

Parameters:

- `token: any` – The token asset.

Return Value Example:

```
2
```

### Methods for Account Management

**`createAccount`**
This method creates an account for a specified user and associated token accounts for fungible or non-fungible tokens. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations. This method can be called only by a `Token Admin` of the chaincode.
A user account has a unique ID, which is formed by an SHA-256 hash of the `orgId` parameter and the `userId` parameter.
A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).
A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this single non-fungible token account.

```
Ctx.ERC1155Account.createAccount(orgId: string, userId: string,
ftAccount: boolean, nftAccount: boolean)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `ftAccount: boolean` – If true, a fungible token account is created and associated with the user account.
- `nftAccount: boolean` – If true, a non-fungible token account is created and associated with the user account.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~cf20877546f52687f387e7c91d88b9722c97e1a456cc0484f40c747f7804feae",
  "userId": "user1",
  "orgId": "appdev",
  "totalAccounts": 2,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
      "tokenId": ""
    }
  ],
  "associatedNftAccount":
"oaccount~73c3e835dac6d0a56ca9d8def08269f83cefd59b9d297fe2cdc5a9083828fa58"
}
```

**`createUserAccount`**
This method creates an account for a specified user. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations.
An account ID is an SHA-256 hash of the `orgId` parameter and the `userId`parameter. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.ERC1155Account.createUserAccount(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON object of the user account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "idcqa",
  "orgId": "appdev",
  "totalAccounts": 0,
```

```
  "totalFtAccounts": 0,
  "associatedFtAccounts": [],
  "associatedNftAccount": ""
}
```

**createTokenAccount**

This method creates a fungible or non-fungible token account to associate with a user account.

A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).

A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this single non-fungible token account.

This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.ERC1155Account.createTokenAccount(orgId: string, userId: string,
tokenType: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenType: TokenType` – The type of token account to create. The only supported token types are `nonfungible` and `fungible`.

Returns:

- On success, a JSON object of the token account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
  "userId": "idcqa",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
```

```
      "tokenId": ""
    }
  ],
  "associatedNftAccount": ""
}
```

**associateTokenToToken**
This method associates a user's fungible token account to a particular fungible token.

```
Ctx.ERC1155Account.associateTokenToToken(accountId: string, tokenId: string)
```

Parameters:

- `accountId: string` – The user account ID.
- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON object of the user account, which shows that the fungible token was associated to the token account. For example, in the following example, the first object in the `associatedFtAccounts` array shows that the fungible token account ID and the token ID are associated.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "idcqa",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "tokenId": "tokenOne"
    }
  ],
  "associatedNftAccount": ""
}
```

**getAccountHistory**
This method returns history for a specified token account.

```
Ctx.ERC1155Account.getAccountHistory(accountId: string)
```

Parameters:

- `accountId: string` – The user account ID.

Returns:

- On success, an array of JSON objects that describes the account history.

Return Value Example:


```
[
    {
        "trxId":
"a2cfc6fc064334d6b9931cdf67193711ec2ff5c50a4714f11855fe7384f00e35",
        "timeStamp": "2023-06-06T14:44:31.000Z",
        "value": {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "assetType": "oaccount",
            "bapAccountVersion": 1,
            "balance": 100,
            "orgId": "appdev",
            "tokenId": "loy1",
            "tokenName": "loyalty",
            "tokenType": "fungible",
            "userId": "idcqa"
        }
    },
    {
        "trxId":
"de483cf7505ae4e7018c4b604c3ab9327c2fb1f802d9408e22735667c1d6997f",
        "timeStamp": "2023-06-06T14:43:23.000Z",
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion": 0,
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "userId": "idcqa",
            "orgId": "appdev",
            "tokenType": "fungible",
            "tokenId": "loy1",
            "tokenName": "loyalty",
            "balance": 0
        }
    },
    {
        "trxId":
"db053e653d3ad9aa5b7b6e04b7cd51aacfbb413272d857a155b60d2a6a12bf4d",
        "timeStamp": "2023-06-05T16:59:08.000Z",
        "value": {
            "assetType": "oaccount",
            "bapAccountVersion": 0,
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "userId": "idcqa",
```

```
            "orgId": "appdev",
            "tokenType": "fungible",
            "tokenId": "",
            "balance": 0
        }
    }
]
```

**getAccountWithStatus**

This method returns token account details, including account status, for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC1155Account.GetAccountWithStatus(accountId, tokenId...)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `tokenId?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details, including the account status.

Return Value Example (Non-Fungible Token Account):

```
{
    "assetType": "oaccount",
    "bapAccountVersion": 1,
    "status": "active",
    "accountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
    "userId": "idcqa",
    "orgId": "appdev",
    "tokenType": "nonfungible",
    "noOfNfts": 1
  }
```

Return Value Example (Fungible Token Account):

```
{
  "bapAccountVersion": 0,
  "assetType": "oaccount",
  "status": "active",
  "accountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479ed4",
  "userId": "idcqa",
  "orgId": "appdev",
  "tokenType": "fungible",
```

```
  "tokenId": "t1",
  "tokenName": "loyalty",
  "balance": 0
}
```

**`getAccount`**

This method returns token account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC1155Account.getAccount(userAccountId: string, tokenId: string)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `tokenId?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details. The `bapAccountVersion` parameter is defined in the account object for internal use.

Return Value Example (Non-Fungible Token Account):

```
{
    "assetType": "oaccount",
    "bapAccountVersion": 0,
    "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
    "userId": "idcqa",
    "orgId": "appdev",
    "tokenType": "nonfungible",
    "noOfNfts": 3
}
```

Return Value Example (Fungible Token Account):

```
{
    "assetType": "oaccount",
    "bapAccountVersion": 0,
    "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
    "userId": "idcqa",
    "orgId": "appdev",
    "tokenType": "fungible",
    "tokenId": "loy1",
    "tokenName": "loyalty",
    "balance": 50
}
```

**getAllAccounts**

This method returns details of all user accounts.

```
Ctx.ERC1155Account.getAllAccounts()
```

Parameters:

- none

Return Value Example:


```
[
        {
            "assetType": "ouaccount",
            "accountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
            "userId": "idcqa",
            "orgId": "appdev",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371"
        },
        {
            "assetType": "ouaccount",
            "accountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
            "userId": "user1_minter",
            "orgId": "appdev",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId":
"oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446"
        },
    ]
```

**`getAccountDetailsByUser`**

This method returns an account summary for a specified user and details of fungible and non-fungible tokens that are associated with the user..

```
Ctx.ERC1155Account.getAccountDetailsByUser(orgId: string, userId:
string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes and account summary for the specified user and details of fungible and non-fungible tokens that are associated with the user. For fractional non-fungible tokens, the `tokenShare` property in the `associatedNFTs` section shows the share that the user owns

Return Value Example:

```
{
    "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
    "associatedFTAccounts": [
        {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "tokenId": "FT",
            "balance": 50
        }
    ],
    "associatedNFTAccount": {
        "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
        "associatedNFTs": [
            {
                "nftTokenId": "FNFT",
                "tokenShare": 100
            },
            {
                "nftTokenId": "FNFT2",
                "tokenShare": 110
            },
            {
                "nftTokenId": "NFT"
            }
        ]
```

```
    }
}
```

**getUserByAccountId**
This method returns the user details of a specified account ID.

```
Ctx.ERC1155Account.getUserByAccountId(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`orgId` and `userId`).

Return Value Example:

```
{
    "orgId": "appdev",
    "userId": "idcqa"
}
```

### Methods for Role Management

**AddRoleMember**
This method adds a role to a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.addRoleMember(role: string, userAccountId: string, token:
any)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `role: string` – The name of the role to add to the specified user.
- `token: any` – The token asset.

Returns:

- On success, a message with account details.

Return Value Example:

```
{
  "msg": "Successfully added role 'minter' to Account Id:
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(Org-Id: appdev, User-Id: idcqa)"
}
```

**isInRole**

This method returns a Boolean value to indicate if a user has a specified role. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.isInRole(role: string, userAccountId: string, token:
any)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `role: string` – The name of the role to search for.
- `token: any` – The token asset.

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d2
2a (Org-Id: appdev, User-Id: idcqa) has minter role"
}
```

**removeRoleMember**

This method removes a role from a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.removeRoleMember(role: string, userAccountId: string,
token: any)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `role: string` – The name of the role to remove from the specified user.
- `token: any` – The token asset.

Return Value Example:

```
{
  "msg": "Successfully removed role 'minter' from Account Id:
oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc50
4b (Org-Id: appdev, User-Id: user1)"
}
```

**getAccountsByRole**
This method returns a list of all account IDs for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.getAccountsByRole(role: string, token: any)
```

Parameters:

- `role: string` – The name of the role to search for.
- `token: any` – The token asset.

Return Value Example:

```
{
  "accounts": [

"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",

"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b"
  ]
}
```

**getUsersByRole**
This method returns a list of all users for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.getUsersByRole(role: string, token: any)
```

Parameters:

- `role: string` – The name of the role to search for.
- `token: any` – The token asset.

Return Value Example:

```
{
    "users": [
        {
            "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "orgId": "appdev",
            "userId": "idcqa"
        },
        {
            "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
            "orgId": "appdev",
            "userId": "user1"
        }
```

```
    ]
}
```

### Methods for Transaction History Management

**`getAccountTransactionHistory`**
This method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner. For non-fungible tokens, this method can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Account.getAccountTransactionHistory(accountId: string)
```

Parameters:

- `accountId: string` – The token account ID.

Return Value Example:

```
[
    {
        "transactionId":
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddf
ae140bc~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475
b",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 10,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "transactionType": "DEBIT",
        "balance": 90
    },
    {
        "transactionId":
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddf
ae140bc~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc11
4",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "transactionType": "DEBIT"
    },
```

```
    {
        "transactionId":
"otransaction~c369929e28e78de06c72d020f1418c9a154a7dd280b2e22ebb4ea4485e24912
4~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "timestamp": "2023-06-06T14:47:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~114a1bc78d04be48ee6dc140c32c042ee9481cb118959626f090eec74452242
2~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "timestamp": "2023-06-05T17:17:57.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 100,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT",
        "balance": 100
    }
]
```

**getTransactionById**

This method returns the transaction details for a specified transaction ID.

```
Ctx.ERC1155Transaction.getTransactionById(transactionId: string)
```

Parameters:

- `transactionId: string` – The ID of the transaction.

Return Value Example:

```
{
  "transactionId":
"otransaction~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cb
e~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a53",
  "history": [
    {
      "trxId":
"9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cbe",
      "timeStamp": "2022-12-08T09:01:28.000Z",
      "value": {
        "assetType": "otransaction",
        "transactionId":
"otransaction~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cb
```

```
e~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a53",
        "tokenId": "tokenOne",
        "fromAccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
        "toAccountId": "",
        "transactionType": "BURN",
        "amount": 5,
        "timestamp": "2022-12-08T09:01:28.000Z",
        "triggeredByUserAccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc"
      }
    }
  ]
}
```

**deleteTransactions**
This method deletes transactions before a specified time stamp from the state database.

```
Ctx.ERC1155Transaction.deleteTransactions(referenceTime: Date)
```

Parameters:

- `referenceTime: Date` – All transactions before this time stamp will be deleted.

Return Value Example:

```
{
  "msg": "Successfuly deleted transaction older than date: Thu Apr 07
2022 21:18:59 GMT+0000 (Coordinated Universal Time).",
  "transactions": [

"otransaction~30513757d8b647fffaafac440d743635f5c1b2e41b25ebd6b70b5bbf7
8a2643f",

"otransaction~ac0e908c735297941ba58bb208ee61ff4816a1e54c090d68024f82adf
743892b"
  ]
}
```

### Methods for Token Behavior Management - Mintable Behavior

**mintBatch**
This method creates (mints) multiple tokens in a batch operation. This method creates only fungible tokens or fractional non-fungible tokens.
For fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. You cannot mint more than the `max_mint_quantity` property of the token, if that property was specified when the token was created or updated.
For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to

mint tokens. Additionally, the caller must also be the creator of the token. There is no upper limit to the quantity of fractional non-fungible tokens that can be minted.
You cannot use this method to mint a whole non-fungible token.

```
Ctx.ERC1155Token.mintBatch(accountId: string, tokenIds: string[],
quantities: number[])
```

Parameters:

- `accountId: string` – The account ID of the user.
- `tokenIds: string[]` – The list of token IDs to mint tokens for.
- `quantity: number[]` – The list of quantities of tokens to mint, corresponding to the token ID array.

Returns:

- On success, a JSON object that includes details on the minted tokens.

Return Value Example:

```
{
    "msg": "Successfully minted batch of tokens for User-Account-Id
ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38
(Org-Id: appdev, User-Id: idcqa).",
    "details": [
        {
            "msg": "Successfully minted 100 tokens of fractional tokenId:
plot55 to Org-Id: appdev, User-Id: idcqa"
        },
        {
            "msg": "Successfully minted 100 tokens of tokenId: loyalty to
Token-Account-Id
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e"
        }
    ]
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`batchTransferFrom`**
This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
This method does not validate that the caller of the method is the specified sender.

```
Ctx.ERC1155Token.batchTransferFrom(fromUserAccountId: string,
toUserAccountId: string, tokenIds: string[], quantities: number[])
```

Parameters:

- `fromUserAccountId: string` – The account ID of the sender and token owner in the current organization.
- `toUserAccountId: string` – The account ID of the receiver.
- `tokenIds: string[]` – A list of token IDs for the tokens to transfer.
- `quantity: number[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10'
quantity from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a100973
71 (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f59034
46 (Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from
Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d41
0e (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f62823797
4c (Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-
Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a100973
71 (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f59034
46 (Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**`safeBatchtransferFrom`**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.

The caller of the method must be the specified sender.

```
Ctx.ERC1155Token.safeBatchTransferFrom(fromUserAccountId: string,
toUserAccountId: string, tokenIds: string[], quantities: number[])
```

Parameters:

- `fromUserAccountId: string` – The account ID of the sender and token owner in the current organization.
- `toUserAccountId: string` – The account ID of the receiver.
- `tokenIds: string[]` – A list of token IDs for the tokens to transfer.
- `quantity: number[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**`balanceOfBatch`**

This method completes a batch operation that gets the balance of token accounts. The account details are specified in three separate lists of organization IDs, user IDs, and token

IDs. This method can be called only by a `Token Admin` of the chaincode or by account owners. Account owners can see balance details only for accounts that they own.

```
Ctx.ERC1155Account.balanceOfBatch(accountIds: string[], tokenIds:
string[])
```

Parameters:

- `accountIds: string[]` – A list of the user account IDs.
- `tokenIds: string[]` – A list of the token IDs.

Return Value Example:

```
[
    {
        "orgId": "appdev",
        "userId": "idcqa",
        "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "tokenAccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
        "tokenId": "FNFT",
        "balance": 100
    },
    {
        "orgId": "appdev",
        "userId": "idcqa",
        "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "tokenAccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
        "tokenId": "FT",
        "balance": 50
    },
    {
        "orgId": "appdev",
        "userId": "user1_minter",
        "userAccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352
003b",
        "tokenAccountId":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "tokenId": "FNFT",
        "balance": 10
    }
]
```

**exchangeToken**
This method exchanges tokens between specified accounts. This method only supports exchanging between an NFT (whole or fractional) and a fungible token or a fungible token and an NFT (whole or fractional). This method can be called only by the account owner.

```
Ctx.ERC1155Token.exchangeToken( fromTokenId: string, fromUserAccountId:
string, fromTokenQuantity: number, toTokenId: string, toUserAccountId:
string, toTokenQuantity: number)
```

Parameters:

- `fromTokenId: string` – The ID of the token that the sender owns.
- `fromUserAccountId: string` – The account ID of the sender.
- `fromTokenQuantity: number` – The quantity of tokens from the sender to exchange with the receiver.
- `toTokenId: string` – The ID of the token that the receiver owns.
- `toUserAccountId: string` – The account ID of the receiver.
- `toTokenQuantity: number` – The quantity of tokens from the receiver to exchange with the sender.

Returns:

- On success, a message with token exchange details.

Return Value Example:

```
{
    "msg": "Succesfully exchanged 10 tokens of type nonfungible with
tokenId: [r1] from Account
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(OrgId: appdev, UserId: idcqa) to 10 tokens of type fungible with tokenId:
[loy1] from Account
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(OrgId: appdev, UserId: user1_minter)"
}
```

### Methods for Token Behavior Management - Burnable Behavior

**burn**
This method deactivates, or burns, the specified fungible and non-fungible tokens.

```
Ctx.ERC1155Token.burn(accountId: string, tokenIds: string[], quantities:
number[])
```

Parameters:

- `accountId: string` – The account ID of the user.
- `tokenIds: string[]` – The list of token IDs to burn.

- `quantity: number[]` – The list of quantities of tokens to burn, corresponding to the token ID array.

Returns:

- On success, a message with details about the burn operations.

Return Value Example:

```
[
  {
    "msg": "Successfully burned NFT token: 'art' from Account-Id:
oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282
c6 (Org-Id: appdev, User-Id: idcqa)"
  },
  {
    "msg": "Successfully burned 5 tokens of tokenId: tokenOne from
Account-ID
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d2
2a (Org-Id: appdev, User-Id: idcqa)"
  },
  {
    "msg": "Successfully burned 2 token share of tokenId: FNFT from
Account-ID
oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac50
4a (Org-Id: AutoF1377358917, User-Id: idcqa)"
  }
]
```

## TypeScript Methods for ERC-1155 NFT Locking

Blockchain App Builder automatically generates methods that you can use to lock non-fungible tokens that use the extended ERC-1155 standard.

A locked token cannot be burned or transferred to other users. All other properties, such as the token's state, owner, and history are preserved. You can use the NFT locking functionality when transferring a token to another blockchain network, such as Ethereum or Polygon.

Before you can lock NFTs, you must assign the vault manager role to a user. The vault manager is a special type of role, a `TokenSys` role. `TokenSys` roles are different from asset-based roles such as burner, minter, and notary, and from administrative roles such as `Token Admin` and `Org Admin`. Currently Blockchain App Builder supports the `vault TokenSys` role. The single user who has the `vault` role for a chaincode is the vault manager of the chaincode, and can manage locked NFTs.

The typical flow for using the NFT locking functionality follows these steps.

- Create a non-fungible token that has the lockable behavior.
- Use the `addTokenSysRole` method to give the `vault` role to a user, the vault manager.
- Call the `lockNFT` method to lock a non-fungible token, specified by the token ID.

### TokenSys Role Management Methods

**`addTokenSysRole`**
This method adds a `TokenSys` role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async addTokenSysRole(orgId: string, userId: string, role: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.addTokenSysRoleMember",
 "TOKEN");
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId);
  return await this.Ctx.ERC1155Token.addTokenSysRoleMember(role,
userAccountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to give to the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully added role 'vault' to Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1)"
}
```

**`isInTokenSysRole`**
This method returns a Boolean value to indicate if a user has a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async isInTokenSysRole(orgId: string, userId: string, role: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.isInTokenSysRole",
"TOKEN", {orgId: orgId, userId: userId });
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId);
  return await this.Ctx.ERC1155Token.isInTokenSysRole(role, userAccountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to check.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1) has vault role"
}
```

**removeTokenSysRole**

This method removes a `TokenSys` role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string())
public async removeTokenSysRole(orgId: string, userId: string, role:
string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.removeTokenSysRol
eMember", "TOKEN");
  const userAccountId =
this.Ctx.ERC1155Account.generateAccountId(orgId, userId);
  return await this.Ctx.ERC1155Token.removeTokenSysRoleMember(role,
userAccountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to remove.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully removed role 'vault' from Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
```

```
ba (Org-Id: Org1MSP, User-Id: user1)"
}
```

**transferTokenSysRole**

This method transfers a `TokenSys` role from a user to another user. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string())
public async transferTokenSysRole(fromOrgId: string, fromUserId: string,
toOrgId: string, toUserId: string, role: string) {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.transferTokenSysRole",
"TOKEN");
    const fromUserAccountId = await
this.Ctx.ERC1155Account.generateAccountId(fromOrgId, fromUserId);
    const toUserAccountId = await
this.Ctx.ERC1155Account.generateAccountId(toOrgId, toUserId);
    return await this.Ctx.ERC1155Token.transferTokenSysRole(role,
fromUserAccountId, toUserAccountId);
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role from.
- `fromUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role from.
- `toOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role to.
- `toUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role to.
- `role: string` – The name of the `TokenSys` role to transfer.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully transfered role 'vault' from Account Id:
ouaccount~f4e311528f03fffa7810753d643f66289ff6c9080fcf839902f28a1d3aff1789
(Org-Id: Org1MSP, User-Id: user1) to Account Id:
ouaccount~ae5be2ae8f98d6d32f5d02b43877d987114e7937c7bacbc30390dcce09996a19
(Org-Id: Org1MSP, User-Id: user2)"
}
```

**getAccountsByTokenSysRole**
This method returns a list of all account IDs for a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string())
public async getAccountsByTokenSysRole(role: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getAccountsByToke
nSysRole", "TOKEN");
  return await this.Ctx.ERC1155Token.getAccountsByTokenSysRole(role);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "accountIds": [

"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03
eba"
    ]
}
```

**getUsersByTokenSysRole**
This method returns user information for all users with a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
@Validator(yup.string())
public async getUsersByTokenSysRole(role: string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getUsersByTokenSy
sRole", "TOKEN");
  return await this.Ctx.ERC1155Token.getUsersByTokenSysRole(role);
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "users":[
      {

"accountId":"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c
2caf03eba",
         "orgId":"Org1MSP",
         "userId":"user1"
      }
   ]
}
```

### NFT Locking Methods

**`lockNFT`**
This method locks a specified non-fungible token. To lock a token, there must be a user with the `TokenSys vault` role, who acts as the vault manager. This method can be called only by the owner of the token.

```
@Validator(yup.string())
public async lockNFT(orgId: string, userId: string, tokenId: string) {
  return await this.Ctx.ERC1155Token.lockNFT(orgId, userId, tokenId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user (optional).
- `tokenID: string` – The ID of the token to lock.

Returns:

- On success, a JSON representation of the token object.

Return Value Example:

```
{
   "assetType":"otoken",
   "tokenId":"token1",
   "tokenName":"artcollection",
   "tokenStandard":"erc1155+",
   "tokenType":"nonfungible",
   "tokenUnit":"whole",
   "behaviors":[
      "indivisible",
      "mintable",
      "transferable",
      "burnable",
```

```
      "lockable",
      "roles"
   ],
   "roles":{
      "minter_role_name":"minter"
   },
   "mintable":{
      "max_mint_quantity":20000
   },
   "quantity":1,

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304
ebff1b6a7733463",
   "creationDate":"2023-10-20T09:16:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff
1b6a7733463",
   "isBurned":false,
   "isLocked":true,
   "tokenUri":"token1.example.com",
   "price":120,
   "on_sale_flag":false
}
```

**isNFTLocked**

This method returns a Boolean value to indicate if a specified token is locked. This method can be called only by the token owner, the vault manager (the user with the `TokenSys vault` role), or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string())
public async isNFTLocked(tokenId: string) {
  try {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.isNFTLocked",
"TOKEN", { tokenId });
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC1155Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VA
ULT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
  const isNFTLocked = await this.Ctx.ERC1155Token.isNFTLocked(tokenId);
  return {isNFTLocked};
}
```

Parameters:

- `tokenID: string` – The ID of the token.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "isNFTLocked":true
}
```

**getAllLockedNFTs**
This method returns a list of all locked non-fungible tokens. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator()
public async getAllLockedNFTs() {
  try {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getAllLockedNFTs",
"TOKEN");
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC1155Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VAULT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
  return this.Ctx.ERC1155Token.getAllLockedNFTs();
}
```

Parameters:

- None

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:

```
[
   {
      "key":"token1",
      "valueJson":{
         "assetType":"otoken",
         "tokenId":"token1",
         "tokenName":"artcollection",
         "tokenStandard":"erc1155+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "mintable",
            "transferable",
```

```
            "burnable",
            "lockable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
         },
         "mintable":{
            "max_mint_quantity":20000
         },
         "quantity":1,

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304
ebff1b6a7733463",
         "creationDate":"2023-10-20T09:16:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff
1b6a7733463",
         "isBurned":false,
         "isLocked":true,
         "tokenUri":"token1.example.com",
         "price":120,
         "on_sale_flag":false
      }
   }
]
```

**getAllLockedNFTsByOrg**

This method returns a list of all locked non-fungible tokens for a specified organization and optionally a specified user. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
@GetMethod()
@Validator(yup.string(), yup.string())
public async getLockedNFTsByOrg(orgId: string, userId?: string) {
  try {
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155TOKEN.getLockedNFTsByOr
g", "TOKEN");
  } catch(err) {
    const isCallerTokenSysRoleHolder = await
this.Ctx.ERC1155Token.isCallerTokenSysRoleHolder(TOKEN_SYS_ROLE_TYPE.VA
ULT);
    if(!isCallerTokenSysRoleHolder)
      throw err;
  }
  return await this.Ctx.ERC1155Token.getLockedNFTsByOrg(orgId, userId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.

- `userId: string` – The user name or email ID of the user (optional).

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:


```
[
   {
      "key":"token1",
      "valueJson":{
         "assetType":"otoken",
         "tokenId":"token1",
         "tokenName":"artcollection",
         "tokenStandard":"erc1155+",
         "tokenType":"nonfungible",
         "tokenUnit":"whole",
         "behaviors":[
            "indivisible",
            "mintable",
            "transferable",
            "burnable",
            "lockable",
            "roles"
         ],
         "roles":{
            "minter_role_name":"minter"
         },
         "mintable":{
            "max_mint_quantity":20000
         },
         "quantity":1,

"createdBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "creationDate":"2023-10-20T09:16:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
         "isBurned":false,
         "isLocked":true,
         "tokenUri":"token1.example.com",
         "price":120,
         "on_sale_flag":false
      }
   }
]
```


## TypeScript Methods for ERC-1155 Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for tokens that use the extended ERC-1155 standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, and transferring tokens. Additionally, other users cannot transfer tokens to a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.



### Automatically Generated Account Status Methods

**`getAccountStatus`**
This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
@Validator(yup.string(), yup.string(), yup.string())
  public async getAccountStatus(orgId: string, userId: string,
tokenId ?: string) {
    const userAccountId =
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
    let tokenAccount = await
this.Ctx.ERC1155Account.getAccount(userAccountId, tokenId);
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT_STATUS.get",
"TOKEN", { accountId: tokenAccount.accountId });
    try {
      return await
this.Ctx.ERC1155AccountStatus.getAccountStatus(tokenAccount.accountId);
    } catch (err) {
      return await
this.Ctx.ERC1155AccountStatus.getDefaultAccountStatus(tokenAccount.acco
untId);
    }
  }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

**`getAccountStatusHistory`**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
public async getAccountStatusHistory(orgId: string, userId: string,
tokenId ?: string) {
    const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
    let tokenAccount = await
this.Ctx.ERC1155Account.getAccount(userAccountId, tokenId);
    await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT_STATUS.history",
"TOKEN", { accountId: tokenAccount.accountId });
    const status_id = await
this.Ctx.ERC1155AccountStatus.generateAccountStatusId(tokenAccount.accountId)
;
    let accountStatusHistory: any;
    try {
      accountStatusHistory = await
this.Ctx.ERC1155AccountStatus.history(status_id);
    } catch (err) {
      return [];
    }
    return accountStatusHistory;
  }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, the account status history in JSON format.

Return Value Example:


```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "suspended"
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "active"
    }
  }
]
```


**activateAccount**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode. Deleted accounts cannot be activated. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
@Validator(yup.string(), yup.string(), yup.string())
public async activateAccount(orgId: string, userId: string, tokenId ?:
string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT_STATUS.activate
Account", "TOKEN");
  const userAccountId =
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
```

```
  let tokenAccount = await this.Ctx.ERC1155Account.getAccount(userAccountId,
tokenId);
  return await
this.Ctx.ERC1155Account.activateAccount(tokenAccount.accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "active"
}
```

**`suspendAccount`**

This method suspends a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
@Validator(yup.string(), yup.string(), yup.string())
public async suspendAccount(orgId: string, userId: string, tokenId ?:
string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT_STATUS.suspendAccount
", "TOKEN");
  const userAccountId = this.Ctx.ERC1155Account.generateAccountId(orgId,
userId, ACCOUNT_TYPE.USER_ACCOUNT);
  let tokenAccount = await this.Ctx.ERC1155Account.getAccount(userAccountId,
tokenId);
  return await
this.Ctx.ERC1155Account.suspendAccount(tokenAccount.accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "status": "suspended"
}
```

**`deleteAccount`**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot be changed to any other state. To delete an account, the account balance must be zero.

```
@Validator(yup.string(), yup.string(), yup.string())
public async deleteAccount(orgId: string, userId: string, tokenId ?:
string) {
  await
this.Ctx.ERC1155Auth.checkAuthorization("ERC1155ACCOUNT_STATUS.deleteAc
count", "TOKEN");
  const userAccountId =
this.Ctx.ERC1155Account.generateAccountId(orgId, userId,
ACCOUNT_TYPE.USER_ACCOUNT);
  let tokenAccount = await
this.Ctx.ERC1155Account.getAccount(userAccountId, tokenId);
  return await
this.Ctx.ERC1155Account.deleteAccount(tokenAccount.accountId);
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

- `tokenId ?: string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

### Account Status SDK Methods

**`getAccountStatus`**
This method gets the current status of the token account.

```
Ctx.ERC1155AccountStatus.getAccountStatus(accountId: string)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

**`getAccountStatusHistory`**
This method gets the history of the account status.

```
Ctx.ERC1155AccountStatus.history(statusId: string)
```

Parameters:

- `statusId: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:

```
[
  {
    "trxId":
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
    "timeStamp": "2022-12-02T10:39:14.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "suspended"
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "status": "active"
    }
  }
]
```

**`activateAccount`**

This method activates a token account. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
Ctx.ERC1155Account.activateAccount(tokenAccountId: string)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "active"
}
```

**`suspendAccount`**
This method suspends a token account.

```
Ctx.ERC1155Account.suspendAccount(tokenAccountId: string)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "suspended"
}
```

**`deleteAccount`**
This method deletes a token account.

```
Ctx.ERC1155Account.deleteAccount(tokenAccountId: string)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
  "status": "deleted"
}
```

## Scaffolded Go Token Project for ERC-1155

Blockchain App Builder takes the input from your token specification file and generates a fully-functional scaffolded chaincode project.

The project automatically generates token lifecycle classes and functions, including CRUD and non-CRUD methods. Validation of arguments, marshalling/unmarshalling, and transparent persistence capability are all supported automatically.

For information on the scaffolded project and methods that are not directly related to tokens, see Scaffolded Go Chaincode Project.

**Reference:**

- Model
- Controller
  - Automatically Generated Token Methods
- SDK Methods

### Model

Transparent Persistence Capability, or simplified ORM, is captured in the `OchainModel` class. The following model shows a whole non-fungible token.

```
package model

type ArtCollection struct {
    AssetType     string `json:"AssetType" final:"otoken"`
    TokenId       string `json:"TokenId" id:"true" mandatory:"true"
validate:"regexp=^[A-Za-z0-9][A-Za-z0-9_-]*$,max=16"`
    TokenName     string `json:"TokenName" final:"artcollection"`
    TokenDesc     string `json:"TokenDesc" validate:"max=256"`
    TokenStandard string `json:"TokenStandard" final:"erc1155+"`
    TokenType     string `json:"TokenType" final:"nonfungible"
validate:"regexp=^nonfungible$"`
    TokenUnit     string `json:"TokenUnit" final:"whole"
validate:"regexp=^whole$"`
```

```
    Mintable map[string]interface{} `json:"Mintable"
final:"{\"Max_mint_quantity\":20000}"`

    Behaviors []string `json:"Behaviors"
final:"[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"burnabl
e\",\"roles\"]"`

    Roles map[string]interface{} `json:"Roles"
final:"{\"minter_role_name\":\"minter\"}"`

    Owner           string `json:"Owner,omitempty" validate:"string"`
    CreatedBy       string `json:"CreatedBy,omitempty" validate:"string"`
    TransferredBy   string `json:"TransferredBy,omitempty" validate:"string"`
    CreationDate    string `json:"CreationDate,omitempty" validate:"string"`
    TransferredDate string `json:"TransferredDate,omitempty"
validate:"string"`
    IsBurned        bool   `json:"IsBurned" validate:"bool"`
    BurnedBy        string `json:"BurnedBy,omitempty" validate:"string"`
    BurnedDate      string `json:"BurnedDate,omitempty" validate:"string"`
    TokenUri        string `json:"TokenUri" mandatory:"true"
validate:"string,max=2000"`

    TokenMetadata ArtCollectionMetadata `json:"TokenMetadata"`

    Price        int  `json:"Price" validate:"int"`
    On_sale_flag bool `json:"On_sale_flag" validate:"bool"`
}

type ArtCollectionMetadata struct {
    Painting_name string `json:"Painting_name" validate:"string"`
    Description   string `json:"Description" validate:"string"`
    Image         string `json:"Image" validate:"string"`
    Painter_name  string `json:"Painter_name" validate:"string"`
}

type Loyalty struct {
    AssetType     string `json:"AssetType" final:"otoken"`
    TokenId       string `json:"TokenId" id:"true" mandatory:"true"
validate:"regexp=^[A-Za-z0-9][A-Za-z0-9_-]*$,max=16"`
    TokenName     string `json:"TokenName" final:"loyalty"`
    TokenDesc     string `json:"TokenDesc" validate:"max=256"`
    TokenStandard string `json:"TokenStandard" final:"erc1155+"`
    TokenType     string `json:"TokenType" final:"fungible"
validate:"regexp=^fungible$"`
    TokenUnit     string `json:"TokenUnit" final:"fractional"
validate:"regexp=^fractional$"`

    Mintable map[string]interface{} `json:"Mintable"
final:"{\"Max_mint_quantity\":10000}"`

    Divisible map[string]interface{} `json:"Divisible"
final:"{\"Decimal\":2}"`

    Behaviors []string `json:"Behaviors"
final:"[\"divisible\",\"mintable\",\"transferable\",\"burnable\",\"roles\"]"`
```

```
    Roles map[string]interface{} `json:"Roles"
final:"{\"minter_role_name\":\"minter\"}"`

    Currency_name           string      `json:"Currency_name"
validate:"string"`
    Token_to_currency_ratio int
`json:"Token_to_currency_ratio" validate:"int"`
    Metadata                interface{} `json:"Metadata,omitempty"`
}
```

The following model shows a fractional non-fungible token.

```
type RealEstateProperty struct {
      AssetType string `json:"AssetType" final:"otoken"`
      TokenId string `json:"TokenId" id:"true" mandatory:"true"
validate:"regexp=^[A-Za-z0-9][A-Za-z0-9_-]*$,max=16"`
      TokenName string `json:"TokenName" final:"realestateproperty"`
      TokenDesc string `json:"TokenDesc" validate:"max=256"`
      TokenStandard string `json:"TokenStandard" final:"erc1155+"`
      TokenType string `json:"TokenType" final:"nonfungible"
validate:"regexp=^nonfungible$"`
      TokenUnit string `json:"TokenUnit" final:"fractional"
validate:"regexp=^fractional$"`

      Mintable map[string]interface{} `json:"Mintable"
final:"{\"Max_mint_quantity\":0}"`
      Behaviors []string `json:"Behaviors"
final:"[\"divisible\",\"mintable\",\"transferable\",\"roles\"]"`

      Divisible map[string]interface{} `json:"Divisible"
final:"{\"Decimal\":0}"`

      Roles map[string]interface{} `json:"Roles"
final:"{\"minter_role_name\":\"minter\"}"`

      CreatedBy string `json:"CreatedBy,omitempty" validate:"string"`
      CreationDate string `json:"CreationDate,omitempty"
validate:"string"`
      IsBurned bool `json:"IsBurned" validate:"bool"`
      TokenUri string `json:"TokenUri" mandatory:"true"
validate:"string,max=2000"`
      Quantity float64 `json:"Quantity,omitempty"`
      TokenMetadata RealEstatePropertyMetadata `json:"TokenMetadata"`

      PropertySellingPrice int `json:"PropertySellingPrice"
validate:"int"`
      PropertyRentingPrice int `json:"PropertyRentingPrice"
validate:"int"`
}

type RealEstatePropertyMetadata struct {
      PropertyType string `json:"PropertyType" validate:"string"`
      PropertyName string `json:"PropertyName" validate:"string"`
```

```
      PropertyAddress string `json:"PropertyAddress" validate:"string"`
      PropertyImage string `json:"PropertyImage" validate:"string"`
}
```

### Controller

There is only one main controller.

```
type Controller struct {
    Ctx trxcontext.TrxContext
}
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable. The other methods are hidden.

You can use the token SDK methods to write custom methods for your business application.

### Automatically Generated Token Methods

Blockchain App Builder automatically generates methods to support tokens and token life cycles. You can use these methods to initialize tokens, manage roles and accounts, and complete other token lifecycle tasks without any additional coding. Controller methods must be public to be invokable. Public method names begin with an upper case character. Method names that begin with a lower case character are private.

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

**`IsTokenAdmin`**
This method returns the Boolean value `true` if the caller of the function is a `Token Admin`, otherwise it returns `false`.This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) IsTokenAdmin(orgId string, userId string) (interface{},
error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Admin.IsUserTokenAdmin",
"TOKEN", map[string]string{"orgId": orgId, "userId": userId})
            if err != nil || !auth {
                  return false, fmt.Errorf("error in authorizing the caller
```

```
%s", err.Error())
            }
            return t.Ctx.ERC1155Auth.IsUserTokenAdmin(orgId, userId)
      }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- The method returns `true` if the caller is a `Token Admin`, otherwise it returns `false`.

Return Value Example:

```
{"result": true}
```

**AddTokenAdmin**

This method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenAdmin(orgId string, userId string)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Admin.AddAdmin", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Admin.AddAdmin(orgId, userId)
      }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{"msg":"Successfully added Admin (orgId: appDev, userId: user1)"}
```

**RemoveTokenAdmin**

This method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. You cannot remove yourself as a `Token Admin`.

```
func (t *Controller) RemoveTokenAdmin(orgId string, userId string)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Admin.RemoveAdmin", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Admin.RemoveAdmin(orgId, userId)
      }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that includes details of the user who was removed as a `Token Admin` of the chaincode.

Return Value Example:

```
{
    "msg": "Successfully removed Admin (orgId appdev userId user1)"
}
```

**GetAllTokenAdmins**

This method returns a list of all users who are a `Token Admin` of the chaincode. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) GetAllTokenAdmins() (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Admin.GetAllAdmins", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Admin.GetAllAdminUsers()
      }
```

Parameters:

- none

Returns:

- On success, an `admins` array in JSON format.

Return Value Example:

```
{
  "admins": [
    {
      "OrgId": "appdev",
      "UserId": "idcqa"
    },
    {
      "OrgId": "appdev",
      "UserId": "user1"
    }
  ]
}
```

### Methods for Token Configuration Management

**`Init`**

This method is called when the chaincode is instantiated. Every `Token Admin` is identified by the `userId` and `orgId` information in the `adminList` parameter. The `userId` is the user name or email ID of the instance owner or the user who is logged in to the instance. The `orgId` is the membership service provider (MSP) ID of the user in the current network organization. The `adminList` parameter is mandatory the first time you deploy the chaincode. If you are upgrading the chaincode, pass an empty list (`[]`). If you are the user who initially deployed the chaincode, you can also specify new admins in the `adminList` parameter when you are upgrading the chaincode. Any other information in the `adminList` parameter is ignored during upgrades.

```
func (t *Controller) Init(adminList
[]erc1155Admin.ERC1155TokenAdminAsset) (interface{}, error) {
      list, err := t.Ctx.ERC1155Admin.InitAdmin(adminList)
      if err != nil {
            return nil, fmt.Errorf("initialising admin list failed
%s", err.Error())
      }
      <1st Token Name> := <1st TokenClassName>{}
      _, err = t.Ctx.ERC1155Token.SaveClassInfo(&<1st Token Name>)
      if err != nil {
            return nil, err
      }
         .
         .
      <nth Token Name> := <nth TokenClassName>{}
      _, err = t.Ctx.ERC1155Token.SaveClassInfo(&<nth Token Name>)
      if err != nil {
            return nil, err
      }
      _, err = t.Ctx.ERC1155Token.SaveDeleteTransactionInfo()
      if err != nil {
            fmt.Println("error: ", err)
```

```
      }
      return list, err
}
```

Parameters:

- `adminList array` – An array of `{OrgId, UserId}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

**`Create<Token Name>Token`**
This method creates tokens. Every token that is defined has its own create method. For fungible tokens, this method can be called only by a `Token Admin` of the chaincode. For non-fungible tokens, if the minter role is defined in the specification file, any user with the minter role can call this method to create an NFT. If the minter role is not defined, any user can use this method to create (mint) NFTs. The user who calls this method becomes the owner of the NFT.

Fungible Tokens:

```
func (t *Controller) Create<Token Name>Token(tokenAsset <Token Class>)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.Save", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Token.Save(&tokenAsset)
    }
```

Non-Fungible Tokens:

```
func (t *Controller) Create<Token Name>Token(tokenAsset <Token Class>,
quantity float64) (interface{}, error) {
         quantityToPass := []float64{quantity}
          return t.Ctx.ERC1155Token.Save(&tokenAsset, quantityToPass...)
    }
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.
- `quantity: number` – For non-fungible tokens only, the number of tokens to mint. The only supported value for this parameter is `1`.

Returns:

- On success, the token asset in JSON format, which includes the following information, depending on the token type.
- `Behaviors` – A list of token behaviors. This property cannot be edited.
- `CreatedBy` – The account ID of the caller, who is the user minting the token. This property cannot be edited.

- `CreationDate` – The time stamp of the minting transaction. This property cannot be edited.
- `IsBurned` – This property indicates whether the token is burned. This property cannot be edited.
- `Mintable` – The properties related to minting. The `max_mint_quantity` value defines the maximum number of tokens that can be created for the token class.
- `Owner` – The account ID of the current owner, who is the caller of the method.
- `Symbol` – The symbol of the token. This property cannot be edited.
- `TokenDesc` – The description of the token.
- `TokenMetadata` – JSON information that describes the token.
- `TokenName` – The name of the token. This property cannot be edited.
- `TokenStandard` – The standard of the token. This property cannot be edited.
- `TokenType` – The type of the token (fungible or non-fungible). This property cannot be edited.
- `TokenUnit` – The unit of the token (whole or fractional). This property cannot be edited.
- `TokenUri` – The URI of the token.
- `Quantity` – The quantity of the token.

Return Value Example (Whole NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-12-29T09:57:03+05:30",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 500
    },
    "OnSaleFlag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "Price": 100,
    "Quantity": 1,
    "Roles": {
```

```
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "token description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "artcollection",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"
}
```


Return Value Example (Fungible Token):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "Currency_name": "Dollar",
    "Divisible": {
        "Decimal": 2
    },
    "Mintable": {
        "Max_mint_quantity": 10000
    },
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "Loyalty",
    "TokenName": "loyalty",
    "TokenStandard": "erc1155+",
    "TokenType": "fungible",
    "TokenUnit": "fractional",
    "Token_to_currency_ratio": 0
}
```

Return Value Example (Fractional NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2023-06-14T09:53:53+05:30",
    "Divisible": {
        "Decimal": 2
    },
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Price": 1000,
    "Quantity": 100,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "realEstate",
    "TokenMetadata": {
        "Description": "Painting Description",
        "Image": "",
        "Painter_name": "",
        "Painting_name": "Paint"
    },
    "TokenName": "realestate",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "fractional",
    "TokenUri": "www.realestate.example.com"
}
```


**Update<Token Name>Token**

This method updates tokens. Every token that is defined has its own update method. You cannot update token metadata or the token URI of non-fungible tokens. For fungible tokens, this method can be called only by a `Token Admin` of the chaincode. For non-fungible tokens, this method can be called only by the token owner.

Fungible Tokens:

```
func (t *Controller) Update<%=tokenModelName%>Token(tokenAsset
<%=tokenModelName%>) (interface{}, error) {
        auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.Update", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Token.Update(&tokenAsset)
    }
```

Non-Fungible Tokens:

```
func (t *Controller) Update<%=tokenModelName%>Token(tokenAsset
<%=tokenModelName%>) (interface{}, error) {
            return t.Ctx.ERC1155Token.Update(&tokenAsset)
    }
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.

Returns:

- On success, the updated token asset in JSON format.

Return Value Example (Whole NFT):

```
{
    "AssetType": "otoken",
    "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2022-12-29T09:57:03+05:30",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 500
    },
    "OnSaleFlag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Price": 100,
    "Quantity": 1,
```

```
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "token description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "artcollection",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"
}
```

**GetTokenHistory**

This method returns the history for a specified token ID. Anyone can call this method.

```
func (t *Controller) GetTokenHistory(tokenId string) (interface{},
error) {
            return t.Ctx.ERC1155Token.GetTokenHistory(tokenId)
      }
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON array that contains the token history.

Return Value Example (Fungible Token):

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-08T09:54:11Z",
    "TxId":
"823sa7c7a00941c62285c86f922bc4d3f5326a20f4bf2f82daa5bc661e4682e8",
    "Value": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
```

```
      "Currency_name": "Rupees",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "Updated Token Description",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-08T09:54:11Z",
    "TxId":
"711bb7c7a00941c62285c86f922bc3b3f5326a20f4bf2f82daa5bc661e4682e8",
    "Value": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "Dollar",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
```

```
  }
]
```

Return Value Example (Fractional NFT):

```
[
    {
        "Timestamp": "2023-06-20T01:06:33Z",
        "TrxId":
"16e53db4f8107f9634b7c3a0a2a81a00f69b634f2a52902b809e544d07f272b1",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "CreationDate": "2023-06-20T01:02:27Z",
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owners": [
                {
                    "AccountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
                    "TokenShare": 10
                },
                {
                    "AccountId":
"oaccount~3cddfdaa855900579d963aa6f755a4aed1f3a474a2462c1b45bd7f36df673
224",
                    "TokenShare": 10
                }
            ],
            "Price": 2000,
            "Quantity": 20,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
```

```
            "TokenId": "FNFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "realestate",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "fractional",
            "TokenUri": "www.FNFT.example.com"
        }
    },
    {
        "Timestamp": "2023-06-20T01:02:27Z",
        "TrxId":
"cec80910d087682554048f911d2cf98b66382bbcf1615483fa1c96c7ea08077c",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "CreationDate": "2023-06-20T01:02:27Z",
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owners": [
                {
                    "AccountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
                    "TokenShare": 20
                }
            ],
            "Price": 2000,
            "Quantity": 20,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "FNFT",
```

```
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "realestate",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "fractional",
            "TokenUri": "www.FNFT.example.com"
        }
    }
]
```


Return Value Example (Whole NFT):


```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-20T01:15:56Z",
        "TxId":
"89a3df3ebbe6dca2bcfbd51fc7dca9aab818a2af746b79a92dc8155b729ab22d",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "CreationDate": "2023-06-20T01:15:56Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owner":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "Price": 2000,
            "Quantity": 1,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "Updated Token Description",
            "TokenId": "NFT",
```

```
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "artcollection",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "www.NFT.example.com"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-20T01:15:56Z",
        "TxId":
"90d6af3ebbe6dca2bcfbd51fc7dca9aab818a2af746b79a92dc8155b729ab22d",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "CreationDate": "2023-06-20T01:15:56Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owner":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "Price": 2000,
            "Quantity": 1,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "NFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "artcollection",
            "TokenStandard": "erc1155+",
```

```
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "www.NFT.example.com"
        }
    }
]
```

**GetAllTokens**

This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAllTokens() (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetAllTokens",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Token.GetAllTokens()
      }
```

Parameters:

- none

Returns:

- A list of all token assets in JSON format.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
```

```
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenTwo",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "art",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
```

```
      ],
      "BurnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "BurnedDate": "2022-12-08T10:49:37Z",
      "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "CreationDate": "2022-12-08T10:45:10Z",
      "IsBurned": true,
      "Mintable": {
        "Max_mint_quantity": 20000
      },
      "OnSaleFlag": false,
      "Owner": "",
      "Price": 0,
      "Roles": {
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "art",
      "TokenMetadata": {
        "Description": "",
        "Image": "",
        "PainterName": "",
        "PaintingName": ""
      },
      "TokenName": "artcollection",
      "TokenStandard": "erc1155+",
      "TokenType": "nonfungible",
      "TokenUnit": "whole",
      "TokenUri": "art.example.com",
      "TransferredBy":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
      "TransferredDate": "2022-12-08T10:47:04Z"
    }
  },
  {
    "key": "FNFT",
    "valueJson": {
        "AssetType": "otoken",
        "Behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
        "CreationDate": "2023-06-20T01:02:27Z",
```

```
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Price": 2000,
            "Quantity": 20,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "FNFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "realestate",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "fractional",
            "TokenUri": "www.FNFT.example.com"
        }
      },
      {
        "key": "FNFT",
        "valueJson": {
            "AssetType": "otoken",
            "Behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
    "oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "CreationDate": "2023-06-20T01:02:27Z",
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Price": 2000,
            "Quantity": 20,
```

```
        "Roles": {
            "burner_role_name": "burner",
            "minter_role_name": "minter"
        },
        "TokenDesc": "",
        "TokenId": "FNFT",
        "TokenMetadata": {
            "Description": "",
            "Image": "",
            "Painter_name": "",
            "Painting_name": ""
        },
        "TokenName": "realestate",
        "TokenStandard": "erc1155+",
        "TokenType": "nonfungible",
        "TokenUnit": "fractional",
        "TokenUri": "www.FNFT.example.com"
    }
  }
]
```

**GetTokenById**

This method returns a token object if the token is present in the state database. For fractional NFTs, the list of owners is also returned. This method can be called only by a `Token Admin` of the chaincode or the token owner.

```
func (t *Controller) GetTokenById(tokenId string) (interface{}, error)
{
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.Get", "TOKEN",
map[string]string{"tokenId": tokenId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Token.GetTokenById(tokenId)
      }
```

Parameters:

- `tokenId string` – The ID of the token to get.

Return Value Example (Whole NFT):

```
{
  "AssetType": "otoken",
  "Behaviors": [
    "indivisible",
    "singleton",
    "mintable",
    "transferable",
    "burnable",
```

```
    "roles"
  ],
  "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "CreationDate": "2022-12-08T10:55:29Z",
  "IsBurned": false,
  "Mintable": {
    "Max_mint_quantity": 20000
  },
  "OnSaleFlag": false,
  "Owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "Price": 0,
  "Quantity": 1,
  "Roles": {
    "minter_role_name": "minter"
  },
  "TokenDesc": "",
  "TokenId": "nftToken",
  "TokenMetadata": {
    "Description": "",
    "Image": "",
    "PainterName": "",
    "PaintingName": ""
  },
  "TokenName": "artcollection",
  "TokenStandard": "erc1155+",
  "TokenType": "nonfungible",
  "TokenUnit": "whole",
  "TokenUri": "nftToken.example.com"
}
```


Return Value Example (Fungible Token):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "Currency_name": "Dollar",
    "Divisible": {
        "Decimal": 2
    },
    "Mintable": {
        "Max_mint_quantity": 10000
    },
    "Roles": {
        "burner_role_name": "burner",
```

```
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "Loyalty",
    "TokenName": "loyalty",
    "TokenStandard": "erc1155+",
    "TokenType": "fungible",
    "TokenUnit": "fractional",
    "Token_to_currency_ratio": 0
}
```


Return Value Example (Fractional NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2023-06-14T09:53:53+05:30",
    "Divisible": {
        "Decimal": 2
    },
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Owners": [
        {
            "AccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "TokenShare": 100
        }
    ],
    "Price": 1000,
    "Quantity": 100,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "realEstate",
    "TokenMetadata": {
        "Description": "Painting Description",
```

```
        "Image": "",
        "Painter_name": "",
        "Painting_name": "Paint"
    },
    "TokenName": "realestate",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "fractional",
    "TokenUri": "www.realestate.example.com"
}
```

**GetAllTokensByUser**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetAllTokensByUser(orgId string, userId string)
(interface{}, error) {
            accountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in GetAllTokensByUser.
Error: %s", err)
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetAllTokensByUser",
"TOKEN", map[string]string{"accountId": accountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Token.GetAllTokensByUser(accountId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
```

```
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "nftToken",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "CreationDate": "2022-12-08T10:55:29Z",
      "IsBurned": false,
      "Mintable": {
        "Max_mint_quantity": 20000
      },
      "OnSaleFlag": false,
      "Owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "Price": 0,
      "Quantity": 1,
      "Roles": {
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
```

```
      "TokenId": "nftToken",
      "TokenMetadata": {
        "Description": "",
        "Image": "",
        "PainterName": "",
        "PaintingName": ""
      },
      "TokenName": "artcollection",
      "TokenStandard": "erc1155+",
      "TokenType": "nonfungible",
      "TokenUnit": "whole",
      "TokenUri": "example.com"
    }
  }
]
```

**`OwnerOf`**

This method returns the account ID, organization ID, and user ID of the owner of the specified token ID. Anyone can call this method.

```
func (t *Controller) OwnerOf(tokenId string) (interface{}, error) {
            return t.Ctx.ERC1155Token.OwnerOf(tokenId)
      }
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example (Whole NFT):

```
{
  "AccountId":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "OrgId": "appdev",
  "UserId": "idcqa"
}
```

Return Value Example (Fractional NFT):

```
[
    {
        "AccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "OrgId": "Org1MSP",
        "UserId": "admin"
    },
    {
        "AccountId":
"oaccount~74108eca702bab6d8548e740254f2cc7955d886885251d52d065042172a59db0",
        "OrgId": "Org1MSP",
        "UserId": "user"
```

```
    }
]
```

**URI**

This method returns the URI of a specified token. Anyone can call this method.

```
func (t *Controller) URI(tokenId string) (interface{}, error) {
            return t.Ctx.ERC1155Token.TokenURI(tokenId)
      }
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{
    "TokenUri": "example.com"
}
```

**Name**

This method returns the name of the token class. Anyone can call this method.

```
func (t *Controller) Name(tokenId string) (interface{}, error) {
            return t.Ctx.ERC1155Token.Name(tokenId)
      }
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{"TokenName": "artcollection"}
```

**TotalSupply**

This method returns the total number of minted tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TotalSupply(tokenDetail erc1155Role.TokenDetail)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.TotalSupply",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            token, err :=
```

```
t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Token.TotalSupply(token)
      }
```

Parameters:

- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"TokenName":"artCollection"}
  ```

Return Value Example:

```
{
    "TotalSupply": 100
}
```

**`TotalNetSupply`**
This method returns the total number of minted tokens minus the number of burned tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TotalNetSupply(tokenDetail erc1155Role.TokenDetail)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.TotalNetSupply", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            token, err := t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Token.TotalNetSupply(token)
      }
```

Parameters:

- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"TokenName":"artCollection"}
  ```

Return Value Example:

```
{
   "TotalNetSupply": 900
}
```

**GetTokensByName**

This method returns all of the token assets for a specified token name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetTokensByName(tokenName string) (interface{},
error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetTokensByName",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Token.GetTokensByName(tokenName)
      }
```

Parameters:

- `tokenName: string` – The name of the token.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
```

```
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenTwo",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  }
]
```

**GetTokenDecimal**

This method returns the number of decimal places for a specified token. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetTokenDecimal(tokenId string) (interface{},
error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetTokenDecimal",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            tokenDecimal, err :=
t.Ctx.ERC1155Token.GetDecimals(tokenId)
            if err != nil {
                  return nil, fmt.Errorf("error in GetTokenDecimal:
%s", err.Error())
            }
            response := make(map[string]interface{})
            response["msg"] = fmt.Sprintf("Token Id: %s has %d decimal
places.", tokenId, tokenDecimal)
            return response, nil
      }
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{
    "msg": "Token Id: tokenOne has 2 decimal places."
}
```

### Methods for Account Management

**CreateAccount**

This method creates an account for a specified user and associated token accounts for fungible or non-fungible tokens. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations. This method can be called only by a `Token Admin` of the chaincode.

A user account has a unique ID, which is formed by an SHA-256 hash of the `orgId` parameter and the `userId` parameter.

A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).

A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId`

parameter, and the constant string `nft` separated by the tilde symbol (~). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this account.
User account IDs start with with `ouaccount~`. Token account IDs start with `oaccount~`.

```
func (t *Controller) CreateAccount(orgId string, userId string, ftAccount
bool, nftAccount bool) (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.CreateAccount", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Account.CreateAccount(orgId, userId,
ftAccount, nftAccount)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `ftAccount bool` – If true, a fungible token account is created and associated with the user account.
- `nftAccount bool` – If true, a non-fungible token account is created and associated with the user account.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
  "AssetType": "ouaccount",
  "AccountId":
"ouaccount~cf20877546f52687f387e7c91d88b9722c97e1a456cc0484f40c747f7804feae",
  "UserId": "user1",
  "OrgId": "appdev",
  "TotalAccounts": 2,
  "TotalFtAccounts": 1,
  "AssociatedFtAccounts": [
    {
      "AccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
      "TokenId": ""
    }
  ],
  "AssociatedNftAccount":
"oaccount~73c3e835dac6d0a56ca9d8def08269f83cefd59b9d297fe2cdc5a9083828fa58"
}
```

**CreateUserAccount**

This method creates an account for a specified user. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations.
An account ID is an SHA-256 hash of the `orgId` parameter and the `userId` parameter. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) CreateUserAccount(orgId string, userId string)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.CreateUserAccount"
, "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Account.CreateUserAccount(orgId,
userId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a JSON object of the user account that was created.

Return Value Example:

```
{
  "AssetType": "ouaccount",
  "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
  "UserId": "idcqa",
  "OrgId": "appdev",
  "TotalAccounts": 0,
  "TotalFtAccounts": 0,
  "AssociatedFtAccounts": [],
  "AssociatedNftAccount": ""
}
```

**CreateTokenAccount**

This method creates a fungible or non-fungible token account to associate with a user account.
A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a

counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (~).
A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (~). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this account.
This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) CreateTokenAccount(orgId string, userId string,
tokenType erc1155Token.TokenType) (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.CreateTokenAccount",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            typeOfToken, err := tokenType.GetTokenType()
          if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Account.CreateTokenAccount(orgId, userId,
typeOfToken)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenType erc1155Token.TokenType` – The type of token account to create. The only supported token types are `nonfungible` and `fungible`.

Returns:

- On success, a JSON object of the token account that was created.

Return Value Example:

```
{
    "AssetType": "ouaccount",
    "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
    "UserId": "idcqa",
    "OrgId": "appdev",
    "TotalAccounts": 1,
    "TotalFtAccounts": 1,
    "AssociatedFtAccounts": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "TokenId": ""
```

```
        }
    ],
    "AssociatedNftAccount": ""
}
```

**AssociateFungibleTokenAccount**

This method associates a user's fungible token account to a particular fungible token. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) AssociateFungibleTokenToAccount(orgId string,
userId string, tokenId string) (interface{}, error) {
            accountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in generating
accountId. Error: %s", err)
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.AssociateFungibleT
okenToAccount", "TOKEN", map[string]string{"accountId": accountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return
t.Ctx.ERC1155Account.AssociateTokenToToken(accountId, tokenId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenId string` – The ID of the token.

Returns:

- On success, a JSON object of the user account, which shows that the fungible token was associated to the token account. For example, in the following example, the first object in the `associatedFtAccounts` array shows that the fungible token account ID and the token ID are associated.

Return Value Example:

```
{
    "AssetType": "ouaccount",
    "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
    "UserId": "idcqa",
```

```
    "OrgId": "appdev",
    "TotalAccounts": 1,
    "TotalFtAccounts": 1,
    "AssociatedFtAccounts": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "TokenId": "tokenOne"
        }
    ],
    "AssociatedNftAccount": ""
}
```

**GetAccountHistory**

This method returns history for a specified token account. This is an asynchronous method. This method can be called only by the `Token Admin` of the chaincode or by the account owner.

```
func (t *Controller) GetAccountHistory(orgId string, userId string,
tokenId ...string) (interface{}, error) {
            userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId, constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in GetAccountHistory. Error:
%s", err)
            }
            _, err =
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.History", "TOKEN",
map[string]string{"accountId": userAccountId})
            if err != nil {
                  return nil, fmt.Errorf("error in GetAccountHistory. Error:
%s", err)
            }
            tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId,
tokenId...)
            if err != nil {
                  return nil, fmt.Errorf("error in GetAccountHistory. Error:
%s", err)
            }
            tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
            if err != nil {
                  return nil, fmt.Errorf("error in GetAccountHistory. Error:
%s", err)
            }
            return t.Ctx.ERC1155Account.GetAccountHistory(tokenAccountId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.

- `userId string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, an array of JSON objects that describes the account history.

Return Value Example:


```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-06T11:03:48Z",
        "TxId":
"c5180f3be3d9130f25a4b4e866f38a4283117dcbfbffb4f55e2c5b03dba0dd29",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "AssetType": "oaccount",
            "Balance": 100,
            "BapAccountVersion": 1
            "OrgId": "appdev",
            "TokenId": "loy1",
            "TokenName": "loyalty",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-06T11:02:39Z",
        "TxId":
"6f81b0c94b451d375a3892446aefbdf78d9fd1ac43699daa89f0dff10db5fd22",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "AssetType": "oaccount",
            "Balance": 0,
            "BapAccountVersion": 0
            "OrgId": "appdev",
            "TokenId": "loy1",
            "TokenName": "loyalty",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
```

```
        "Timestamp": "2023-06-05T16:28:46Z",
        "TxId":
"8185af648546e909488e72149be497b210f74f95ada252c42da9c35cb9d98691",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "AssetType": "oaccount",
            "Balance": 0,
            "BapAccountVersion": 0
            "OrgId": "appdev",
            "TokenId": "",
            "TokenName": "",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```

**GetAccount**

This method returns token account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
func (t *Controller) GetAccount(orgId string, userId string,
tokenId ...string) (interface{}, error) {
            accountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
            if err != nil {
                  return nil, err
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.GetAccount", "TOKEN",
map[string]string{"accountId": accountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            return t.Ctx.ERC1155Account.GetAccountWithStatus(accountId,
tokenId...)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details. The `BapAccountVersion` and `AccountCategory` parameters are defined in the account object for internal use.

Return Value Example (Non-Fungible Token Account):

```
{
  "AccountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419
a9a",
  "AssetType": "oaccount",
  "BapAccountVersion": 1,
  "NoOfNfts": 1,
  "OrgId": "appdev",
  "Status": "active",
  "TokenType": "nonfungible",
  "UserId": "idcqa"
}
```

Return Value Example (Fungible Token Account):

```
{
  "AccountCategory": "",
  "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
  "AssetType": "oaccount",
  "Balance": 0,
  "BapAccountVersion": 0,
  "OrgId": "appdev",
  "Status": "active",
  "TokenId": "t1",
  "TokenName": "loyalty",
  "TokenType": "fungible",
  "UserId": "idcqa"
}
```

**`GetAllAccounts`**

This method returns details of all user accounts. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetAllAccounts() (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.GetAllAccounts",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Account.GetAllAccounts()
      }
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:


```
[
    {
        "AssetType": "ouaccount",
        "AccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "UserId": "idcqa",
        "OrgId": "appdev",
        "TotalAccounts": 3,
        "TotalFtAccounts": 2,
        "AssociatedFtAccounts": [
            {
                "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
                "TokenId": "loy1"
            },
            {
                "AccountId":
"oaccount~58c5a6b09a41befca2a9ea2550439838c4dcf4d8a2a4f7c98e9319cf8479bfc4",
                "TokenId": ""
            }
        ],
        "AssociatedNftAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371"
    },
    {
        "AssetType": "ouaccount",
        "AccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
        "UserId": "user1_minter",
        "OrgId": "appdev",
        "TotalAccounts": 2,
        "TotalFtAccounts": 1,
        "AssociatedFtAccounts": [
            {
                "AccountId":
"oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c",
                "TokenId": "loy1"
            }
        ],
        "AssociatedNftAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446"
    },
]
```

**GetAccountDetailsByUser**

This method returns an account summary for a specified user and details of fungible and non-fungible tokens that are associated with the user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
func (t *Controller) GetAccountDetailsByUser(orgId string, userId
string) (interface{}, error) {
            accountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
            if err != nil {
                  return nil, err
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.GetAccountDetailsB
yUser", "TOKEN", map[string]string{"accountId": accountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return t.Ctx.ERC1155Account.GetAccountDetailsByUser(orgId,
userId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes and account summary for the specified user and details of fungible and non-fungible tokens that are associated with the user. For fractional non-fungible tokens, the `TokenShare` property in the `AssociatedNFTs` section shows the share that the user owns.

Return Value Example:

```
{
    "AssociatedFTAccounts": [
        {
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "Balance": 90,
          "TokenId": "FT"
        },
    ],
    "AssociatedNFTAccount": {
        "AccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
```

```
371",
        "AssociatedNFTs": [
            {
              "NFTTokenId": "FNFT",
                "TokenShare": 230
            },
            {
              "NFTTokenId": "NFT"
            },
            {
              "NFTTokenId": "NFT2"
            }
        ]
    },
    "UserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
}
```

**GetUserByAccountId**
This method returns the user details of a specified account ID. This method can be called by any user.

```
func (t *Controller) GetUserByAccountId(accountId string) (interface{},
error) {
            return t.Ctx.ERC1155Account.GetUserByAccountById(accountId)
      }
```

Parameters:

- `accountId string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`orgId` and `userId`).

Return Value Example:

```
{
    "OrgId": "appdev",
    "UserId": "user2"
}
```

### Methods for Role Management

**AddRole**
This method adds a role to a specified user and token. This method can be called only by a `Token Admin` of the chaincode. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. The specified user must have a token account that

is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
func (t *Controller) AddRole(orgId string, userId string, role string,
tokenDetail erc1155Role.TokenDetail) (interface{}, error) {
            userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
            if err != nil {
                  return nil, err
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.AddRoleMember",
"TOKEN", map[string]string{"accountId": userAccountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            token, err :=
t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Token.AddRoleMember(role,
userAccountId, token)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `role: string` – The name of the role to add to the specified user.
- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"TokenName":"artCollection"}
  ```

Returns:

- On success, a message with account details.

Return Value Example:

```
{
    "msg": "Successfully added role minter to
```

```
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(orgId : appdev, userId : idcqa)"
}
```

**IsInRole**

This method returns a Boolean value to indicate if a user has a specified role. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by the `Token Admin` of the chaincode or the `Account Owner` of the account. The specified user must have a token account that is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
func (t *Controller) IsInRole(orgId string, userId string, role string,
tokenDetail erc1155Role.TokenDetail) (interface{}, error) {
            userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId, constants.UserAccount)
            if err != nil {
                  return nil, err
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.IsInRole", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            token, err := t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            var result bool
            result, err = t.Ctx.ERC1155Token.IsInRole(role, userAccountId,
token)
            if err != nil {
                  return nil, fmt.Errorf("error in IsInRole %s", err.Error())
            }
            response := make(map[string]interface{})
            response["result"] = result
            return response, nil
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `role: string` – The name of the role to search for.

- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"TokenName":"artCollection"}
  ```

Return Value Example:

```
{
    "result": true
}
```

**RemoveRole**

This method removes a role from a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by the `Token Admin` of the chaincode. The specified user must have a token account that is associated with the fungible token, or a non-fungible token account for NFT roles. The specified role must exist in the specification file for the token.

```
func (t *Controller) RemoveRole(orgId string, userId string, role
string, tokenDetail erc1155Role.TokenDetail) (interface{}, error) {
            userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
            if err != nil {
                  return nil, err
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.RemoveRoleMember",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            token, err :=
t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Token.RemoveRoleMember(role,
userAccountId, token)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.

- `userId string` – The user name or email ID of the user.
- `role: string` – The name of the role to remove from the specified user.
- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

  For non-fungible tokens, use the following format:

  ```
  {"TokenName":"artCollection"}
  ```

Return Value Example:

```
{
    "msg": "Successfully removed role 'minter' from Account Id:
oaccount~ec7e4de2f81e3ea071710e07b6ff7d9346e84ef665ca4650885dbe8c3e2bd4c0
(Org-Id: appdev, User-Id: idcqa)"
}
```

**GetAccountsByRole**

This method returns a list of all account IDs for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) GetAccountsByRole(role string, tokenDetail
erc1155Role.TokenDetail) (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Role.GetAccountsByRole",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            token, err := t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Role.GetAccountsByRole(role, token)
      }
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

For non-fungible tokens, use the following format:

```
{"TokenName":"artCollection"}
```

Return Value Example:

```
{
    "accounts": [

"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",

"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b"
    ]
}
```

**GetUsersByRole**

This method returns a list of all users for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name. This method can be called only by the `Token Admin` of the chaincode.

```
func (t *Controller) GetUsersByRole(role string, tokenDetail
erc1155Role.TokenDetail) (interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Role.GetUsersByRole",
"TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            token, err :=
t.Ctx.ERC1155Token.GetTokenByIdOrName(tokenDetail)
            if err != nil {
                  return nil, err
            }
            return t.Ctx.ERC1155Role.GetUsersByRole(role, token)
      }
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetails erc1155Role.TokenDetail` – The details that specify the token. For fungible tokens, use the following format:

  ```
  {"TokenId":"token1"}
  ```

For non-fungible tokens, use the following format:

```
{"TokenName":"artCollection"}
```

Return Value Example:

```
{
    "Users": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "OrgId": "appdev",
            "UserId": "idcqa"
        },
        {
            "AccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
            "OrgId": "appdev",
            "UserId": "user1"
        }
    ]
}
```

### Methods for Transaction History Management

**`GetAccountTransactionHistory`**
This method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner. For non-fungible tokens, this method can only be called when connected to the remote Oracle Blockchain Platform network.

```
func (t *Controller) GetAccountTransactionHistory(orgId string, userId
string, tokenId ...string) (interface{}, error) {
            userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId, constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in
GetAccountTransactionHistory. Error: %s", err)
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.GetAccountTransactionHis
tory", "TOKEN", map[string]string{"accountId": userAccountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId,
tokenId...)
            if err != nil {
                  return nil, fmt.Errorf("error in
GetAccountTransactionHistory. Error: %s", err)
            }
```

```
            tokenAccountId, err :=
util.GetAccountProperty(tokenAccount, constants.AccountId)
            if err != nil {
                  return nil, fmt.Errorf("error in
GetAccountTransactionHistory. Error: %s", err)
            }
            return
t.Ctx.ERC1155Account.GetAccountTransactionHistory(tokenAccountId)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Return Value Example:

```
[
    {
        "Balance": 90,
        "Timestamp": "2023-06-06T11:11:09Z",
        "TokenId": "FNFT",
        "TransactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "TransactedAmount": 10,
        "TransactionId":
"otransaction~0f4d96fbf8fed88ea8a313342897772109l467c701848d595ebc3fffa
88b3657~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475
b",
        "TransactionType": "DEBIT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38"
    },
    {
        "Timestamp": "2023-06-06T11:11:09Z",
        "TokenId": "NFT",
        "TransactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446",
        "TransactionId":
"otransaction~0f4d96fbf8fed88ea8a313342897772109l467c701848d595ebc3fffa
88b3657~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc11
4",
        "TransactionType": "DEBIT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38"
```

```
    },
    {
        "Timestamp": "2023-06-06T11:06:54Z",
        "TokenId": "NFT",
        "TransactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "TransactionId":
"otransaction~6a13667ea3f6edc4c854e85b127526eccb58783f653c348b42a3869f0f29a4f
b~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "TransactionType": "MINT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
    },
    {
        "Balance": 100,
        "Timestamp": "2023-06-05T16:34:33Z",
        "TokenId": "FNFT",
        "TransactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "TransactedAmount": 100,
        "TransactionId":
"otransaction~2bc15de1766d582d821bd8d61756bca02837dc683c0aa61f69657ccd1d95e33
5~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "TransactionType": "MINT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
    }
]
```

**GetTransactionById**

This method returns the transaction details for a specified transaction ID. Anyone can call this method.

```
func (t *Controller) GetTransactionById(transactionId string) (interface{},
error) {
            return t.Ctx.ERC1155Transaction.GetTransactionById(transactionId)
      }
```

Parameters:

- `transactionId: string` – The ID of the transaction.

Return Value Example:

```
{
    "history": [
        {
            "IsDelete": "false",
            "Timestamp": "2022-12-08T10:45:56Z",
            "TxId":
"2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948901aa98",
            "Value": {
```

```
                "Amount": 5,
                "AssetType": "otransaction",
                "FromAccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b",
                "Timestamp": "2022-12-08T10:45:56Z",
                "ToAccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
                "TokenId": "tokenOne",
                "TransactionId":
"otransaction~2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948
901aa98~9c3ce5f21abd98ca018c196086d73a812f2f49dba323f1de4f6867eecfeec8f
f",
                "TransactionType": "TRANSFER",
                "TriggeredByUserAccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc"
            }
        }
    ],
    "transactionId":
"otransaction~2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948
901aa98~9c3ce5f21abd98ca018c196086d73a812f2f49dba323f1de4f6867eecfeec8f
f"
}
```

**DeleteHistoricalTransactions**

This method deletes transactions before a specified time stamp from the state database. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) DeleteHistoricalTransactions(timestamp string)
(interface{}, error) {
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Transaction.DeleteHistoric
alTransactions", "TOKEN")
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the
caller %s", err.Error())
            }
            return
t.Ctx.ERC1155Transaction.DeleteHistoricalTransactions(timestamp)
      }
```

Parameters:

- `timestamp: string` – All transactions before this time stamp will be deleted.

Return Value Example:

```
{
    "Transactions": [
```

```
"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf
7"
    ],
    "msg": "Successfuly deleted transaction older than
date:2022-04-06T08:17:53Z"
}
```

### Methods for Token Behavior Management - Mintable Behavior

**`MintBatch`**

This method creates (mints) multiple tokens in a batch operation. This method creates only fungible tokens or fractional non-fungible tokens.

For fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. You cannot mint more than the `max_mint_quantity` property of the token, if that property was specified when the token was created or updated.

For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. Additionally, the caller must also be the creator of the token. There is no upper limit to the quantity of fractional non-fungible tokens that can be minted.

You cannot use this method to mint a whole non-fungible token.

```
func (t *Controller) MintBatch(orgId string, userId string, tokenIds
[]string, quantity []float64) (interface{}, error) {
            accountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in generating the accountId.
Error: %s", err)
            }
            var tokens []interface{}
            for _, tokenId := range tokenIds {
                  tokenAssetValue, err := t.getTokenObject(tokenId)
                  if err != nil {
                        return nil, err
                  }
                  tokens = append(tokens, tokenAssetValue.Interface())
            }
            return t.Ctx.ERC1155Token.MintBatch(accountId, tokens, quantity)
      }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenIds []string` – The list of token IDs to mint tokens for.
- `quantity []float64` – The list of quantities of tokens to mint, corresponding to the token ID array.

Returns:

- On success, a JSON object that includes details on the minted tokens.

Return Value Example:

```
{
    "details": [
        {
            "msg": "Successfully minted 100 tokens of fractional
tokenId: plot55 to Org-Id: appdev, User-Id: idcqa"
        },
        {
            "msg": "Successfully minted 100 tokens of tokenId:
'loyalty' to Token-Account-Id
'oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e'"
        }
    ],
    "msg": "Successfully minted batch of tokens for User-Account-Id
'ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38' (Org-Id: 'appdev', User-Id: 'idcqa')"
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`BatchTransferFrom`**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
This method does not validate that the caller of the method is the specified sender. This method can be called by any user.

```
func (t *Controller) BatchTransferFrom(fromOrgId string, fromUserId
string, toOrgId string, toUserId string, tokenIds []string, quantity
[]float64) (interface{}, error) {
            fromAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(fromOrgId, fromUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            toAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(toOrgId, toUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
```

```
            }
            var tokens []interface{}
            for _, tokenId := range tokenIds {
                  tokenAssetValue, err := t.getTokenObject(tokenId)
                  if err != nil {
                        return nil, err
                  }
                  tokens = append(tokens, tokenAssetValue.Interface())
            }
            return t.Ctx.ERC1155Token.BatchTransferFrom(fromAccountId,
toAccountId, tokens, quantity)
      }
```

Parameters:

- `fromOrgId string` – The membership service provider (MSP) ID of the sender and token owner in the current organization.
- `fromUserId string` – The user name or email ID of the sender and token owner.
- `toOrgId string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId string` – The user name or email ID of the receiver.
- `tokenIds string[]` – A list of token IDs for the tokens to transfer.
- `quantity float64[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
```

```
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f59034
46 (Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**SafeBatchTransferFrom**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
The caller of the method must be the specified sender. This method can be called by any user.

```
func (t *Controller) SafeBatchTransferFrom(fromOrgId string,
fromUserId string, toOrgId string, toUserId string, tokenIds []string,
quantity []float64) (interface{}, error) {
            fromAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(fromOrgId, fromUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            toAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(toOrgId, toUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            var tokens []interface{}
            for _, tokenId := range tokenIds {
                  tokenAssetValue, err := t.getTokenObject(tokenId)
                  if err != nil {
                        return nil, err
                  }
                  tokens = append(tokens, tokenAssetValue.Interface())
            }
            return
t.Ctx.ERC1155Token.SafeBatchTransferFrom(fromAccountId, toAccountId,
tokens, quantity)
      }
```

Parameters:

- `fromOrgId string` – The membership service provider (MSP) ID of the sender and token owner in the current organization.
- `fromUserId string` – The user name or email ID of the sender and token owner.

- `toOrgId string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId string` – The user name or email ID of the receiver.
- `tokenIds string[]` – A list of token IDs for the tokens to transfer.
- `quantity float64[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**`BalanceOfBatch`**

This method completes a batch operation that gets the balance of token accounts. The account details are specified in three separate lists of organization IDs, user IDs, and token IDs. This method can be called only by a `Token Admin` of the chaincode or by account owners. Account owners can see balance details only for accounts that they own.

```
func (t *Controller) BalanceOfBatch(orgIds []string, userIds []string,
tokenIds []string) (interface{}, error) {
            callerAccountCheck := false
            _, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.BalanceOfBatch",
"TOKEN")
            if err != nil {
```

```
                  callerAccountCheck = true
            }
            accountIds, err :=
t.Ctx.ERC1155Account.GenerateAccountIds(orgIds, userIds,
callerAccountCheck)
            if err != nil {
                  return nil, fmt.Errorf("error in BalanceOfBatch.
Error: %s", err)
            }
            var tokens []interface{}
            for _, tokenId := range tokenIds {
                  tokenAssetValue, err := t.getTokenObject(tokenId)
                  if err != nil {
                        tokens = append(tokens, map[string]interface{}
{"TokenId": tokenId})
                  } else {
                        tokens = append(tokens,
tokenAssetValue.Interface())
                  }
            }
            return t.Ctx.ERC1155Account.BalanceOfBatch(accountIds,
tokens)
      }
```

Parameters:

- `orgIds []string` – A list of the membership service provider (MSP) IDs in the current organization.
- `userIds []string` – A list of the user name or email IDs.
- `tokenIds []string` – A list of the token IDs.

Return Value Example:
In the following example, the token ID `FNFT` represents a fractional non-fungible token and the token ID `FT` represents a fungible token.

```
[
    {
        "OrgId": "appdev",
        "UserId": "idcqa",
        "UserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38",
        "TokenAccountId":
"oaccount~e88276a3be547e31b567346bddddde52d37734da4d5fae83ab2e5c98a10097
371",
        "TokenId": "FNFT",
        "Balance": 100
    },
    {
        "OrgId": "appdev",
        "UserId": "idcqa",
        "UserAccountId":
```

```
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "TokenAccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
        "TokenId": "FT",
        "Balance": 50
    },
    {
        "OrgId": "appdev",
        "UserId": "user1_minter",
        "UserAccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
        "TokenAccountId":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "TokenId": "FNFT",
        "Balance": 10
    }
]
```

**ExchangeToken**

This method exchanges tokens between specified accounts. This method only supports exchanging between an NFT and a fungible token or a fungible token and an NFT. This method can be called only by the account owner.

```
func (t *Controller) ExchangeToken(fromTokenId string, fromOrgId string,
fromUserId string, fromTokenQuantity float64, toTokenId string, toOrgId
string, toUserId string, toTokenQuantity float64) (interface{}, error) {
            fromUserAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(fromOrgId, fromUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Account.ExchangeToken",
"TOKEN", map[string]string{"accountId": fromUserAccountId})
            if err != nil && !auth {
                  return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
            }
            toUserAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(toOrgId, toUserId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            return t.Ctx.ERC1155Token.ExchangeToken(fromTokenId,
fromUserAccountId, fromTokenQuantity, toTokenId, toUserAccountId,
toTokenQuantity)
      }
```

Parameters:

- `fromTokenId string` – The ID of the token that the sender owns.
- `fromOrgId string` – The membership service provider (MSP) ID of the sender in the current organization.
- `fromUserId string` – The user name or email ID of the sender.
- `fromTokenQuantity float64` – The quantity of tokens from the sender to exchange with the receiver.
- `toTokenId string` – The ID of the token that the receiver owns.
- `toOrgId string` – The membership service provider (MSP) ID of the receiver in the current organization.
- `toUserId string` – The user name or email ID of the receiver.
- `toTokenQuantity float64` – The quantity of tokens from the receiver to exchange with the sender.

Returns:

- On success, a message with token exchange details.

Return Value Example:

```
{
    "msg": "Succesfully exchanged 10 tokens of type 'nonfungible' with
tokenId: [r1] from Account
'oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371' (OrgId: appdev, UserId: idcqa) to 10 tokens of type 'fungible'
with tokenId: [loy1] from Account
'oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f6282379
74c' (OrgId: 'appdev', UserId: 'user1_minter')"
}
```

### Methods for Token Behavior Management - Burnable Behavior

**`BurnBatch`**

This method deactivates, or burns, the specified fungible and non-fungible tokens. Any user can call this method.

```
func (t *Controller) BurnBatch(orgId string, userId string, tokenIds
[]string, quantity []float64) (interface{}, error) {
            accountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
            if err != nil {
                  return nil, fmt.Errorf("error in BatachTransferFrom.
Error: %s", err)
            }
            var tokens []interface{}
            for _, tokenId := range tokenIds {
                  tokenAssetValue, err := t.getTokenObject(tokenId)
                  if err != nil {
                        return nil, err
```

```
                }
                tokens = append(tokens, tokenAssetValue.Interface())
            }
            return t.Ctx.ERC1155Token.Burn(accountId, tokens, quantity)
        }
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID in the current organization.
- `userId string` – The user name or email ID.
- `tokenIds []string` – The list of the token IDs to burn
- `quantity []float64` – The list of quantities of tokens to burn, corresponding to the token ID array..

Returns:

- On success, a message with details about the burn operations.

Return Value Example:

```
[
    {
        "msg": "Successfully burned NFT token: 'art' from Account-Id:
oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6
(Org-Id: appdev, User-Id: idcqa)"
    },
    {
        "msg": "Successfully burned 2 tokens of tokenId: FT from Account-ID
oaccount~9a940587fd322ccc8400233244cd3b13f3aa2a52e418e4c71fb819a2217bc49e
(Org-Id: AutoF1377358917, User-Id: idcqa)"
    },
    {
        "msg": "Successfully burned 2 token share of tokenId: FNFT from
Account-ID
oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a
(Org-Id: AutoF1377358917, User-Id: idcqa)"
    }
]
```

**BurnNFT**

This method deactivates, or burns, the specified non-fungible token, and returns a token object and token history. Any user with the burner role can call this method.

```
func (t *Controller) BurnNFT(orgId string, userId string, tokenId string)
(interface{}, error) {
            tokenAsset, err := t.Ctx.ERC1155Token.Get(tokenId)
            if err != nil {
                return nil, err
            }
            token := tokenAsset.(map[string]interface{})
            if token[constants.TokenType] != constants.NonFungible {
```

```
                  return nil, fmt.Errorf("only nonfungible tokens are
allowed")
            }
            if token[constants.IsBurned] == true {
                  return nil, fmt.Errorf("token with tokenId %s is
already burned", tokenId)
            }
            tokenQuantity := float64(1)
            tokenUnit := token[constants.TokenUnit]
            tokenHistory, err := t.Ctx.ERC1155Token.History(tokenId)
            if err != nil {
                  return nil, err
            }
            if tokenUnit == constants.Fractional {
                  owners, err := t.Ctx.ERC1155Token.OwnerOf(tokenId)
                  if err != nil {
                        return nil, err
                  }
                  ownersList := owners.([]map[string]interface{})
                  if len(ownersList) != 1 {
                        return nil, fmt.Errorf("NFT has multiple
owners, to completely burn this NFT it should have only one owner")
                  }
                  tokenQuantity = token[constants.Quantity].(float64)
            }
            token[constants.TokenId], err =
strconv.Atoi(token[constants.TokenId].(string))
            if err != nil {
                  return nil, fmt.Errorf("tokenId is expected to be
integer but found %s", tokenId)
            }
            tokenHistoryBytes, err := json.Marshal(tokenHistory)
            if err != nil {
                  return nil, err
            }
            var tokenHistoryAsRawJson json.RawMessage
            err = json.Unmarshal(tokenHistoryBytes,
&tokenHistoryAsRawJson)
            if err != nil {
                  return nil, err
            }
            token[constants.TokenHistory] =
string(tokenHistoryAsRawJson)
            tokenIds := []string{tokenId}
            tokenQuantities := []float64{tokenQuantity}
            token[constants.IsBurned] = true
            _, err = t.BurnBatch(orgId, userId, tokenIds,
tokenQuantities)
            if err != nil {
                  return nil, err
            }
            return token, nil
      }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID in the current organization.
- `userId: string` – The user name or email ID.
- `tokenId: string` – The ID of the non-fungible token to burn

Returns:

- On success, a token object in JSON format that includes token history information.

Return Value Example:


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2023-08-22T13:14:46+05:30",
    "IsBurned": true,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "Price": 120,
    "Quantity": 1,
    "Roles": {
        "minter_role_name": "minter"
    },
    "TokenDesc": "",
    "TokenHistory":
"[{\"IsDelete\":\"false\",\"Timestamp\":\"2023-08-22T13:14:46+05:30\",\"TxId\
":\"c0ea212f19197c5b86323bfca11c6ca545aa0af5d40cd04f9e955b5371fd40ae\",\"Valu
e\":{\"AssetType\":\"otoken\",\"Behaviors\":
[\"indivisible\",\"singleton\",\"mintable\",\"transferable\",\"burnable\",\"r
oles\"],\"CreatedBy\":\"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436
dd1aae201d347ad1288d\",\"CreationDate\":\"2023-08-22T13:14:46+05:30\",\"IsBur
ned\":false,\"Mintable\":
{\"Max_mint_quantity\":20000},\"On_sale_flag\":false,\"Owner\":\"oaccount~42e
89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d\",\"Price\":120
,\"Quantity\":1,\"Roles\":
{\"minter_role_name\":\"minter\"},\"TokenDesc\":\"\",\"TokenId\":\"1\",\"Toke
nMetadata\":
{\"Description\":\"\",\"Image\":\"\",\"Painter_name\":\"\",\"Painting_name\":
\"\"},\"TokenName\":\"artcollection\",\"TokenStandard\":\"erc1155+
```

```
\",\"TokenType\":\"nonfungible\",\"TokenUnit\":\"whole\",\"TokenUri\":\
"example.com\"}}]",
    "TokenId": 1,
    "TokenMetadata": {
        "Description": "",
        "Image": "",
        "Painter_name": "",
        "Painting_name": ""
    },
    "TokenName": "artcollection",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "example.com"
}
```

### SDK Methods

- Access Control Management
- Token Configuration Management
- Account Management
- Role Management
- Transaction History Management
- Token Behavior Management
  - Mintable Behavior
  - Transferable Behavior
  - Burnable Behavior

### Methods for Access Control Management

**`CheckAuthorization`**
Use this method to add an access control check to an operation. This is an asynchronous function. Most automatically generated methods include access control. Certain token methods can be run only by the `ERC721Admin` or `Account Owner` of the token or by a `MultipleAccountOwner` for users with multiple accounts. The `CheckAuthorization` method is part of the `erc721Auth` class, which you access via the `Ctx` object. The access control mapping is described in the `oChainUtil.go` file, as shown in the following text. You can modify access control by editing the `oChainUtil.go` file.

```
  var t TokenAccess
      var r RoleAccess
      var a AccountAccess
      var as AccountStatusAccess
      var h HoldAccess
      var ad AdminAccess
      var trx TransactionAccess
      var tc TokenConversionAccess
      var auth AuthAccess
```

```
        var erc721ad ERC721AdminAccess
        var erc721t ERC721TokenAccess
        var erc721r ERC721RoleAccess
        var erc721a ERC721AccountAccess
        var erc721as ERC721AccountStatusAccess
        var erc721trx ERC721TransactionAccess

        var erc1155ad ERC1155AdminAccess
        var erc1155t ERC1155TokenAccess
        var erc1155a ERC1155AccountAccess
        var erc1155as ERC1155AccountStatusAccess
        var erc1155trx ERC1155TransactionAccess
        var erc1155role ERC1155RoleAccess
        trx.DeleteHistoricalTransactions = []string{"Admin"}
        ad.AddAdmin = []string{"Admin"}
        ad.RemoveAdmin = []string{"Admin"}
        ad.GetAllAdmins = []string{"Admin", "OrgAdmin"}
        ad.AddOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
        ad.RemoveOrgAdmin = []string{"Admin", "OrgAdminOrgIdCheck"}
        ad.GetOrgAdmins = []string{"Admin", "OrgAdmin"}
        ad.IsTokenAdmin = []string{"Admin", "MultipleAccountOwner", "OrgAdmin"}
        t.Save = []string{"Admin"}
        t.GetAllTokens = []string{"Admin", "OrgAdmin"}
        t.Update = []string{"Admin"}
        t.GetTokenDecimals = []string{"Admin", "OrgAdmin"}
        t.GetTokensByName = []string{"Admin", "OrgAdmin"}
        t.AddRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
        t.RemoveRoleMember = []string{"Admin", "OrgAdminRoleCheck"}
        t.IsInRole = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
        t.GetTotalMintedTokens = []string{"Admin", "OrgAdmin"}
        t.GetNetTokens = []string{"Admin", "OrgAdmin"}
        t.Get = []string{"Admin", "OrgAdmin"}
        t.GetTokenHistory = []string{"Admin", "OrgAdmin"}
        a.CreateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
        a.AssociateToken = []string{"Admin", "OrgAdminAccountIdCheck"}
        a.GetAllAccounts = []string{"Admin"}
        a.GetAllOrgAccounts = []string{"Admin", "OrgAdminOrgIdCheck"}
        a.GetAccount = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
        a.History = []string{"Admin", "AccountOwner"}
        a.GetAccountTransactionHistory = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
        a.GetAccountTransactionHistoryWithFilters = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
        a.GetSubTransactionsById = []string{"Admin", "TransactionInvoker"}
        a.GetSubTransactionsByIdWithFilters = []string{"Admin",
"TransactionInvoker"}
        a.GetAccountBalance = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
        a.GetAccountOnHoldBalance = []string{"Admin",
"OrgAdminAccountIdCheck", "AccountOwner"}
        a.GetOnHoldIds = []string{"Admin", "OrgAdminAccountIdCheck",
```

```
"AccountOwner"}
      a.GetAccountsByUser = []string{"Admin", "OrgAdminOrgIdCheck",
"MultipleAccountOwner"}

      as.Get = []string{"Admin", "OrgAdminAccountIdCheck",
"AccountOwner"}
      as.ActivateAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      as.SuspendAccount = []string{"Admin", "OrgAdminOrgIdCheck"}
      as.DeleteAccount = []string{"Admin", "OrgAdminOrgIdCheck"}

      r.GetAccountsByRole = []string{"Admin"}
      r.GetUsersByRole = []string{"Admin"}
      r.GetOrgAccountsByRole = []string{"Admin", "OrgAdminOrgIdCheck"}
      r.GetOrgUsersByRole = []string{"Admin", "OrgAdminOrgIdCheck"}
      tc.InitializeExchangePoolUser = []string{"Admin"}
      tc.AddConversionRate = []string{"Admin"}
      tc.UpdateConversionRate = []string{"Admin"}
      tc.GetConversionRate = []string{"Admin", "OrgAdmin",
"AnyAccountOwner"}
      tc.GetConversionRateHistory = []string{"Admin", "OrgAdmin",
"AnyAccountOwner"}
      tc.TokenConversion = []string{"Admin", "AnyAccountOwner"}
      tc.GetExchangePoolUser = []string{"Admin"}

      erc721ad.AddAdmin = []string{"Admin"}
      erc721ad.GetAllAdmins = []string{"Admin"}
      erc721ad.IsTokenAdmin = []string{"Admin"}
      erc721ad.RemoveAdmin = []string{"Admin"}
      erc721trx.DeleteHistoricalTransactions = []string{"Admin"}
      erc721t.Save = []string{"Admin"}
      erc721t.GetAllTokens = []string{"Admin"}
      erc721t.Update = []string{"Admin"}
      erc721t.GetTokensByName = []string{"Admin"}
      erc721t.AddRoleMember = []string{"Admin"}
      erc721t.RemoveRoleMember = []string{"Admin"}
      erc721t.IsInRole = []string{"Admin", "AccountOwner"}
      erc721t.Get = []string{"Admin", "TokenOwner"}
      erc721t.GetAllTokensByUser = []string{"Admin", "AccountOwner"}
      erc721t.TotalSupply = []string{"Admin"}
      erc721t.TotalNetSupply = []string{"Admin"}
      erc721t.History = []string{"Admin"}
      erc721a.CreateAccount = []string{"Admin"}
      erc721a.CreateUserAccount = []string{"Admin"}
      erc721a.CreateTokenAccount = []string{"Admin"}
      erc721a.AssociateFungibleTokenToAccount = []string{"Admin",
"AccountOwner"}
      erc721a.GetAllAccounts = []string{"Admin"}
      erc721a.History = []string{"Admin", "AccountOwner"}
      erc721a.GetAccountTransactionHistory = []string{"Admin",
"AccountOwner"}
      erc721a.GetAccountTransactionHistoryWithFilters =
[]string{"Admin", "AccountOwner"}
      erc721a.GetAccountByUser = []string{"Admin",
```

```
"MultipleAccountOwner"}
      erc721a.BalanceOf = []string{"Admin", "MultipleAccountOwner"}

      erc721as.Get = []string{"Admin", "AccountOwner"}
      erc721as.ActivateAccount = []string{"Admin"}
      erc721as.SuspendAccount = []string{"Admin"}
      erc721as.DeleteAccount = []string{"Admin"}

      erc721r.GetAccountsByRole = []string{"Admin"}
      erc721r.GetUsersByRole = []string{"Admin"}

      erc1155ad.AddAdmin = []string{"Admin"}
      erc1155ad.GetAllAdmins = []string{"Admin"}
      erc1155ad.IsUserTokenAdmin = []string{"Admin"}
      erc1155ad.RemoveAdmin = []string{"Admin"}
      erc1155t.AddRoleMember = []string{"Admin"}
      erc1155t.IsInRole = []string{"Admin"}
      erc1155t.GetAllTokens = []string{"Admin"}
      erc1155t.GetAllTokensByUser = []string{"Admin", "AccountOwner"}
      erc1155t.Get = []string{"Admin", "TokenOwner"}
      erc1155t.RemoveRoleMember = []string{"Admin"}
      erc1155t.TotalNetSupply = []string{"Admin"}
      erc1155t.TotalSupply = []string{"Admin"}
      erc1155t.GetTokenDecimal = []string{"Admin"}
      erc1155t.GetTokensByName = []string{"Admin"}
      erc1155t.GetTotalMintedTokens = []string{"Admin"}
      erc1155t.GetNetTokens = []string{"Admin"}
      erc1155t.Save = []string{"Admin"}
      erc1155t.Update = []string{"Admin"}

      erc1155trx.DeleteHistoricalTransactions = []string{"Admin"}

      erc1155role.GetAccountsByRole = []string{"Admin"}
      erc1155role.GetUsersByRole = []string{"Admin"}
      erc1155a.AssociateFungibleTokenToAccount = []string{"Admin",
"AccountOwner"}
      erc1155a.BalanceOfBatch = []string{"Admin"}
      erc1155a.CreateAccount = []string{"Admin"}
      erc1155a.CreateTokenAccount = []string{"Admin"}
      erc1155a.CreateUserAccount = []string{"Admin"}
      erc1155a.GetAccountTransactionHistory = []string{"Admin",
"AccountOwner"}
      erc1155a.GetAccountTransactionHistoryWithFilters = []string{"Admin",
"AccountOwner"}
      erc1155a.GetAccountsByUser = []string{"Admin", "AccountOwner"}
      erc1155a.GetAccount = []string{"Admin", "AccountOwner"}
      erc1155a.History = []string{"Admin", "AccountOwner"}
      erc1155a.GetAllAccounts = []string{"Admin"}
      erc1155a.ExchangeToken = []string{"AccountOwner"}
      erc1155a.GetAccountDetailsByUser = []string{"Admin", "AccountOwner"}

      erc1155as.Get = []string{"Admin", "AccountOwner"}
      erc1155as.ActivateAccount = []string{"Admin"}
```

```
erc1155as.SuspendAccount = []string{"Admin"}
erc1155as.DeleteAccount = []string{"Admin"}
var accessMap TokenAccessControl
accessMap.Token = t
accessMap.Account = a
accessMap.AccountStatus = as
accessMap.Hold = h
accessMap.Role = r
accessMap.Admin = ad
accessMap.Auth = auth
accessMap.TokenConversion = tc
accessMap.ERC721ADMIN = erc721ad
accessMap.ERC721TOKEN = erc721t
accessMap.ERC721ACCOUNT = erc721a
accessMap.ERC721AccountStatus = erc721as
accessMap.ERC721ROLE = erc721r
accessMap.ERC721TRANSACTION = erc721trx
accessMap.ERC1155Account = erc1155a
accessMap.ERC1155AccountStatus = erc1155as
accessMap.ERC1155Admin = erc1155ad
accessMap.ERC1155Token = erc1155t
accessMap.ERC1155Transaction = erc1155trx
accessMap.ERC1155Role = erc1155role
```

```
Ctx.ERC1155Auth.CheckAuthorization(funcName string, args []string)
(bool, error)
```

Parameters:

- `funcName string` – The map value between the receivers and methods as described in the `oChainUtil.go` file.
- `args` – A variable argument where `args[0]` takes the constant `'TOKEN'` and `args[1]` takes the `accountId` parameter to add an access control check for an `AccountOwner`. To add an access control check for a `MultipleAccountOwner`, `args[1]` takes the `orgId` parameter and `args[2]` takes the `userId` parameter. To add an access control check for a `TokenOwner`, `args[1]` takes the token ID parameter.

Returns:

- A `bool` response of error as required.

**`IsUserTokenAdmin`**

This method returns the Boolean value `true` if the specified user is a `Token Admin`, and `false` otherwise. The method can be called only by a `Token Admin` of the token chaincode.

```
Ctx.ERC1155Auth.IsUserTokenAdmin(orgId string, userId string)
(interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
{
  "result": true
}
```

**AddAdmin**

This method adds a user as a `Token Admin` of the token chaincode. The method can be called only by a `Token Admin` of the token chaincode.

```
Ctx.ERC1155Admin.AddAdmin(orgId string, userId string) (interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user added as a `Token Admin` of the token chaincode.

Return Value Example:

```
{
  "msg": "Successfully added Admin (orgId: appdev, userId: user1)"
}
```

**RemoveAdmin**

This method removes a user as a `Token Admin` of the token chaincode. The method can be called only by a `Token Admin` of the token chaincode. You cannot remove yourself as a `Token Admin`.

```
Ctx.ERC1155Admin.RemoveAdmin(orgId string, userId string) (interface{},
error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.

Returns:

- On success, a message that lists details for the user removed as a `Token Admin` of the token chaincode.

Return Value Example:

```
{
    "msg": "Successfully removed Admin (orgId appdev userId user1)"
}
```

**GetAllAdminUsers**
This method returns a list of all `Token Admin` users.

```
Ctx.ERC1155Admin.GetAllAdminUsers() (interface{}, error)
```

Parameters:

- none

Returns:

- On success, a list of all `Token Admin` users, identifed by organization ID and user ID.

Return Value Example:

```
{
  "admins": [
    {
      "OrgId": "appdev",
      "UserId": "idcqa"
    },
    {
      "OrgId": "appdev",
      "UserId": "user1"
    }
  ]
}
```

### Methods for Token Configuration Management

**Save**
This method creates tokens. Every token that is defined has its own create method. For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method to create an NFT. If not, any user can use this method to create (mint) NFTs. The user who calls this method becomes the owner of the NFT (whole or fractional).

```
Ctx.ERC1155Token.Save(token interface{}, quantity ...float64)
(interface{}, error)
```

Parameters:

- `tokenAsset: interface{}` – The token asset. The properties of the asset are defined in the model file.

- `quantity: number` – For non-fungible tokens only, the number of tokens to mint. The only supported value for this parameter is `1`.

Returns:

- On success, the token asset in JSON format, which includes the following information.
- `Behavior` or `Behaviors` – A list of token behaviors. This property cannot be edited.
- `CreatedBy` – The account ID of the caller, who is the user minting the token. This property cannot be edited.
- `CreationDate` – The time stamp of the minting transaction. This property cannot be edited.
- `IsBurned` – This property indicates whether the token is burned. This property cannot be edited.
- `Mintable` – The properties related to minting. The `max_mint_quantity` value defines the maximum number of tokens that can be created for the token class.
- `Owner` – The account ID of the current owner, who is the caller of the method.
- `Symbol` – The symbol of the token. This property cannot be edited.
- `TokenDesc` – The description of the token.
- `TokenMetadata` – JSON information that describes the token.
- `TokenName` – The name of the token. This property cannot be edited.
- `TokenStandard` – The standard of the token. This property cannot be edited.
- `TokenType` – The type of the token (fungible or non-fungible). This property cannot be edited.
- `TokenUnit` – The unit of the token (whole or fractional). This property cannot be edited.
- `TokenUri` – The URI of the token.
- `Quantity` – The quantity of the token.

Return Value Example (Whole NFT):

```
{
    "AssetType": "otoken",
    "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2022-12-29T09:57:03+05:30",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 500
```

```
    },
    "OnSaleFlag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "Price": 100,
    "Quantity": 1,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "token description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
    "TokenName": "artcollection",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\
\ .ipfs.infura-ipfs.io/?filename=MonaLisa.jpeg"
}
```


Return Value Example (Fungible Token):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "Currency_name": "Dollar",
    "Divisible": {
        "Decimal": 2
    },
    "Mintable": {
        "Max_mint_quantity": 10000
    },
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "Loyalty",
```

```
    "TokenName": "loyalty",
    "TokenStandard": "erc1155+",
    "TokenType": "fungible",
    "TokenUnit": "fractional",
    "Token_to_currency_ratio": 0
}
```


Return Value Example (Fractional NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2023-06-14T09:53:53+05:30",
    "Divisible": {
        "Decimal": 2
    },
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Price": 1000,
    "Quantity": 100,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "realEstate",
    "TokenMetadata": {
        "Description": "Painting Description",
        "Image": "",
        "Painter_name": "",
        "Painting_name": "Paint"
    },
    "TokenName": "realestate",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "fractional",
    "TokenUri": "www.realestate.example.com"
}
```

**Update**

This method updates tokens. You cannot update token metadata or the token URI of non-fungible tokens.

```
Ctx.ERC1155Token.Update(tokenAsset interface{}) (interface{}, error)
```

Parameters:

- `tokenAsset: interface{}` – The token asset. The properties of the asset are defined in the model file.

Returns:

- On success, the updated token asset in JSON format.

Return Value Example (Whole NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2022-12-29T09:57:03+05:30",
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 500
    },
    "OnSaleFlag": false,
    "Owner":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "Price": 100,
    "Quantity": 1,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "token description",
    "TokenId": "monalisa",
    "TokenMetadata": {
        "Description": "Mona Lisa Painting",
        "Image": "monalisa.jpeg",
        "PainterName": "Leonardo_da_Vinci",
        "PaintingName": "Mona_Lisa"
    },
```

```
    "TokenName": "artcollection",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "whole",
    "TokenUri": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\\\\ .ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"
}
```

**History (Token)**

This method returns the history for a specified token ID.

```
Ctx.ERC1155Token.History(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId string` – The ID of the token.

Returns:

- On success, a JSON array that contains the token history.

Return Value Example (Fungible Token):

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-08T09:54:11Z",
    "TxId":
"823sa7c7a00941c62285c86f922bc4d3f5326a20f4bf2f82daa5bc661e4682e8",
    "Value": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "Rupees",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "Updated Token Description",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
```

```
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-08T09:54:11Z",
    "TxId":
"711bb7c7a00941c62285c86f922bc3b3f5326a20f4bf2f82daa5bc661e4682e8",
    "Value": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "Dollar",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  }
]
```


Return Value Example (Fractional NFT):


```
[
    {
        "Timestamp": "2023-06-20T01:06:33Z",
        "TrxId":
"16e53db4f8107f9634b7c3a0a2a81a00f69b634f2a52902b809e544d07f272b1",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "divisible",
                "mintable",
```

```
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "CreationDate": "2023-06-20T01:02:27Z",
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owners": [
                {
                    "AccountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
                    "TokenShare": 10
                },
                {
                    "AccountId":
"oaccount~3cddfdaa855900579d963aa6f755a4aed1f3a474a2462c1b45bd7f36df673224",
                    "TokenShare": 10
                }
            ],
            "Price": 2000,
            "Quantity": 20,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "FNFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "realestate",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "fractional",
            "TokenUri": "www.FNFT.example.com"
        }
    },
    {
        "Timestamp": "2023-06-20T01:02:27Z",
        "TrxId":
"cec80910d087682554048f911d2cf98b66382bbcf1615483fa1c96c7ea08077c",
        "Value": {
```

```
            "AssetType": "otoken",
            "Behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "CreationDate": "2023-06-20T01:02:27Z",
            "Divisible": {
                "Decimal": 2
            },
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owners": [
                {
                    "AccountId":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
                    "TokenShare": 20
                }
            ],
            "Price": 2000,
            "Quantity": 20,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "FNFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "realestate",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "fractional",
            "TokenUri": "www.FNFT.example.com"
        }
    }
]
```

Return Value Example (Whole NFT):


```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-20T01:15:56Z",
        "TxId":
"89a3df3ebbe6dca2bcfbd51fc7dca9aab818a2af746b79a92dc8155b729ab22d",
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "CreationDate": "2023-06-20T01:15:56Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owner":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
            "Price": 2000,
            "Quantity": 1,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "Updated Token Description",
            "TokenId": "NFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "artcollection",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "www.NFT.example.com"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-20T01:15:56Z",
        "TxId":
"90d6af3ebbe6dca2bcfbd51fc7dca9aab818a2af746b79a92dc8155b729ab22d",
```

```
        "Value": {
            "AssetType": "otoken",
            "Behaviors": [
                "indivisible",
                "singleton",
                "mintable",
                "transferable",
                "roles"
            ],
            "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "CreationDate": "2023-06-20T01:15:56Z",
            "IsBurned": false,
            "Mintable": {
                "Max_mint_quantity": 20000
            },
            "On_sale_flag": true,
            "Owner":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
            "Price": 2000,
            "Quantity": 1,
            "Roles": {
                "burner_role_name": "burner",
                "minter_role_name": "minter"
            },
            "TokenDesc": "",
            "TokenId": "NFT",
            "TokenMetadata": {
                "Description": "",
                "Image": "",
                "Painter_name": "",
                "Painting_name": ""
            },
            "TokenName": "artcollection",
            "TokenStandard": "erc1155+",
            "TokenType": "nonfungible",
            "TokenUnit": "whole",
            "TokenUri": "www.NFT.example.com"
        }
    }
]
```


**GetAllTokens**

This method returns all of the token assets that are saved in the state database. This method can be called only by a `Token Admin` of the chaincode. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.GetAllTokens()() (interface{}, error)
```

Parameters:

- none

Return Value Example:


```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "tokenTwo",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
        "Max_mint_quantity": 1000
      },
      "Roles": {
```

```
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenTwo",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "art",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "BurnedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "BurnedDate": "2022-12-08T10:49:37Z",
      "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "CreationDate": "2022-12-08T10:45:10Z",
      "IsBurned": true,
      "Mintable": {
        "Max_mint_quantity": 20000
      },
      "OnSaleFlag": false,
      "Owner": "",
      "Price": 0,
      "Roles": {
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "art",
      "TokenMetadata": {
        "Description": "",
        "Image": "",
        "PainterName": "",
        "PaintingName": ""
      },
      "TokenName": "artcollection",
      "TokenStandard": "erc1155+",
      "TokenType": "nonfungible",
```

```
      "TokenUnit": "whole",
      "TokenUri": "art.example.com",
      "TransferredBy":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
      "TransferredDate": "2022-12-08T10:47:04Z"
    }
  },
  {
    "key": "FNFT",
    "valueJson": {
        "AssetType": "otoken",
        "Behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a",
        "CreationDate": "2023-06-20T01:02:27Z",
        "Divisible": {
            "Decimal": 2
        },
        "IsBurned": false,
        "Mintable": {
            "Max_mint_quantity": 20000
        },
        "On_sale_flag": true,
        "Price": 2000,
        "Quantity": 20,
        "Roles": {
            "burner_role_name": "burner",
            "minter_role_name": "minter"
        },
        "TokenDesc": "",
        "TokenId": "FNFT",
        "TokenMetadata": {
            "Description": "",
            "Image": "",
            "Painter_name": "",
            "Painting_name": ""
        },
        "TokenName": "realestate",
        "TokenStandard": "erc1155+",
        "TokenType": "nonfungible",
        "TokenUnit": "fractional",
        "TokenUri": "www.FNFT.example.com"
    }
  },
  {
    "key": "FNFT",
    "valueJson": {
```

```
        "AssetType": "otoken",
        "Behaviors": [
            "divisible",
            "mintable",
            "transferable",
            "burnable",
            "roles"
        ],
        "CreatedBy":
"oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac5
04a",
        "CreationDate": "2023-06-20T01:02:27Z",
        "Divisible": {
            "Decimal": 2
        },
        "IsBurned": false,
        "Mintable": {
            "Max_mint_quantity": 20000
        },
        "On_sale_flag": true,
        "Price": 2000,
        "Quantity": 20,
        "Roles": {
            "burner_role_name": "burner",
            "minter_role_name": "minter"
        },
        "TokenDesc": "",
        "TokenId": "FNFT",
        "TokenMetadata": {
            "Description": "",
            "Image": "",
            "Painter_name": "",
            "Painting_name": ""
        },
        "TokenName": "realestate",
        "TokenStandard": "erc1155+",
        "TokenType": "nonfungible",
        "TokenUnit": "fractional",
        "TokenUri": "www.FNFT.example.com"
    }
  }
]
```


**Get (Token)**

This method returns a token object if the token is present in the state database. This method can be called only by a `Token Admin` of the chaincode or the token owner. For fractional NFTs, the list of owners is also returned.

```
Ctx.ERC1155Token.Get(id string, result ...interface{}) (interface{},
error)
```

Parameters:

- `id string` – The ID of the token to get.

Return Value Example (Whole NFT):


```
{
  "AssetType": "otoken",
  "Behaviors": [
    "indivisible",
    "singleton",
    "mintable",
    "transferable",
    "burnable",
    "roles"
  ],
  "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "CreationDate": "2022-12-08T10:55:29Z",
  "IsBurned": false,
  "Mintable": {
    "Max_mint_quantity": 20000
  },
  "OnSaleFlag": false,
  "Owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "Price": 0,
  "Quantity": 1,
  "Roles": {
    "minter_role_name": "minter"
  },
  "TokenDesc": "",
  "TokenId": "nftToken",
  "TokenMetadata": {
    "Description": "",
    "Image": "",
    "PainterName": "",
    "PaintingName": ""
  },
  "TokenName": "artcollection",
  "TokenStandard": "erc1155+",
  "TokenType": "nonfungible",
  "TokenUnit": "whole",
  "TokenUri": "example.com"
}
```


Return Value Example (Fungible Token):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
```

```
            "burnable",
            "roles"
        ],
        "Currency_name": "Dollar",
        "Divisible": {
            "Decimal": 2
        },
        "Mintable": {
            "Max_mint_quantity": 10000
        },
        "Roles": {
            "burner_role_name": "burner",
            "minter_role_name": "minter"
        },
        "TokenDesc": "Token Description",
        "TokenId": "Loyalty",
        "TokenName": "loyalty",
        "TokenStandard": "erc1155+",
        "TokenType": "fungible",
        "TokenUnit": "fractional",
        "Token_to_currency_ratio": 0
}
```


Return Value Example (Fractional NFT):


```
{
    "AssetType": "otoken",
    "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
    "CreationDate": "2023-06-14T09:53:53+05:30",
    "Divisible": {
        "Decimal": 2
    },
    "IsBurned": false,
    "Mintable": {
        "Max_mint_quantity": 20000
    },
    "On_sale_flag": false,
    "Owners": [
        {
            "AccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad12
88d",
            "TokenShare": 100
```

```
        }
    ],
    "Price": 1000,
    "Quantity": 100,
    "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
    },
    "TokenDesc": "Token Description",
    "TokenId": "realEstate",
    "TokenMetadata": {
        "Description": "Painting Description",
        "Image": "",
        "Painter_name": "",
        "Painting_name": "Paint"
    },
    "TokenName": "realestate",
    "TokenStandard": "erc1155+",
    "TokenType": "nonfungible",
    "TokenUnit": "fractional",
    "TokenUri": "www.realestate.example.com"
}
```

**GetAllTokensByUser**

This method returns all of the token assets that are owned by a specified user. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.GetAllTokensByUser(accountId string) (interface{}, error)
```

Parameters:

- `accountId string` – The account ID of the user.

Return Value Example:

```
[
  {
    "key": "tokenOne",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "divisible",
        "mintable",
        "transferable",
        "roles"
      ],
      "Currency_name": "",
      "Divisible": {
        "Decimal": 2
      },
      "Mintable": {
```

```
        "Max_mint_quantity": 1000
      },
      "Roles": {
        "burner_role_name": "burner",
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "tokenOne",
      "TokenName": "moneytok",
      "TokenStandard": "erc1155+",
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  },
  {
    "key": "nftToken",
    "valueJson": {
      "AssetType": "otoken",
      "Behaviors": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
      ],
      "CreatedBy":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "CreationDate": "2022-12-08T10:55:29Z",
      "IsBurned": false,
      "Mintable": {
        "Max_mint_quantity": 20000
      },
      "OnSaleFlag": false,
      "Owner":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a428
2c6",
      "Price": 0,
      "Quantity": 1,
      "Roles": {
        "minter_role_name": "minter"
      },
      "TokenDesc": "",
      "TokenId": "nftToken",
      "TokenMetadata": {
        "Description": "",
        "Image": "",
        "PainterName": "",
        "PaintingName": ""
      },
      "TokenName": "artcollection",
```

```
      "TokenStandard": "erc1155+",
      "TokenType": "nonfungible",
      "TokenUnit": "whole",
      "TokenUri": "example.com"
    }
  }
]
```

**OwnerOf**

This method returns the account ID, organization ID, and user ID of the owner of the specified token ID.

```
Ctx.ERC1155Token.OwnerOf(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{
  "AccountId":
"oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6",
  "OrgId": "appdev",
  "UserId": "idcqa"
}
```

**TokenURI**

This method returns the URI of a specified token. Anyone can call this method.

```
Ctx.ERC1155Token.TokenURI(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{
    "TokenUri": "example.com"
}
```

**Name**

This method returns the name of the token class. Anyone can call this method.

```
Ctx.ERC1155Token.Name(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId string` – The ID of the token.

Return Value Example:

```
{"TokenName": "artcollection"}
```

**TotalSupply**
This method returns the total number of minted tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.TotalSupply(tokenAsset interface{})
(map[string]interface{}, error)
```

Parameters:

- `tokenAsset interface{}` – The token asset.

Return Value Example:

```
{"TotalSupply": 100}
```

**TotalNetSupply**
This method returns the total number of minted tokens minus the number of burned tokens. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.TotalNetSupply(token interface{}) (interface{}, error)
```

Parameters:

- `token interface{}` – The token asset.

Return Value Example:

```
{"TotalNetSupply": 100}
```

**GetTokensByName**
This method returns all of the token assets for a specified token name. This method uses Berkeley DB SQL rich queries and can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Token.GetTokensByName(tokenName string) (interface{}, error)
```

Parameters:

- `tokenName: string` – The name of the token.

Return Value Example:

```
[
  {
    "key": "tokenOne",
```

```
  "valueJson": {
    "AssetType": "otoken",
    "Behaviors": [
      "divisible",
      "mintable",
      "transferable",
      "roles"
    ],
    "Currency_name": "",
    "Divisible": {
      "Decimal": 2
    },
    "Mintable": {
      "Max_mint_quantity": 1000
    },
    "Roles": {
      "burner_role_name": "burner",
      "minter_role_name": "minter"
    },
    "TokenDesc": "",
    "TokenId": "tokenOne",
    "TokenName": "moneytok",
    "TokenStandard": "erc1155+",
    "TokenType": "fungible",
    "TokenUnit": "fractional",
    "Token_to_currency_ratio": 0
  }
},
{
  "key": "tokenTwo",
  "valueJson": {
    "AssetType": "otoken",
    "Behaviors": [
      "divisible",
      "mintable",
      "transferable",
      "roles"
    ],
    "Currency_name": "",
    "Divisible": {
      "Decimal": 2
    },
    "Mintable": {
      "Max_mint_quantity": 1000
    },
    "Roles": {
      "burner_role_name": "burner",
      "minter_role_name": "minter"
    },
    "TokenDesc": "",
    "TokenId": "tokenTwo",
    "TokenName": "moneytok",
    "TokenStandard": "erc1155+",
```

```
      "TokenType": "fungible",
      "TokenUnit": "fractional",
      "Token_to_currency_ratio": 0
    }
  }
]
```

**GetDecimals**

This method returns the number of decimal places for a specified token. If the divisible behavior is not specified for the token, then the default value of zero decimal places is returned.

```
Ctx.ERC1155Token.GetDecimals(tokenId string) (int, error)
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
2
```

### Methods for Account Management

**CreateAccount**

This method creates an account for a specified user and associated token accounts for fungible or non-fungible tokens. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations. This method can be called only by a `Token Admin` of the chaincode.

A user account has a unique ID, which is formed by an SHA-256 hash of the `orgId` parameter and the `userId` parameter.

A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).

A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this single non-fungible token account.

```
Ctx.ERC1155Account.CreateAccount(orgId string, userId string,
ftAccount bool, nftAccount bool) (ERC1155UserAccount, error)
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.

- `userId string` – The user name or email ID of the user.
- `ftAccount bool` – If true, a fungible token account is created and associated with the user account.
- `nftAccount bool` – If true, a non-fungible token account is created and associated with the user account.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
  "AssetType": "ouaccount",
  "AccountId":
"ouaccount~cf20877546f52687f387e7c91d88b9722c97e1a456cc0484f40c747f7804feae",
  "UserId": "user1",
  "OrgId": "appdev",
  "TotalAccounts": 2,
  "TotalFtAccounts": 1,
  "AssociatedFtAccounts": [
    {
      "AccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
      "TokenId": ""
    }
  ],
  "AssociatedNftAccount":
"oaccount~73c3e835dac6d0a56ca9d8def08269f83cefd59b9d297fe2cdc5a9083828fa58"
}
```

**`CreateUserAccount`**

This method creates an account for a specified user. An account must be created for any user who will have tokens at any point. The user account tracks the NFT account and the fungible token accounts that a user has. Users must have accounts in the network to complete token-related operations.
An account ID is an SHA-256 hash of the `orgId` parameter and the `userId` parameter. This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.ERC1155Account.CreateUserAccount(orgId string, userId string)
(interface{}, error)
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a JSON object of the user account that was created.

Return Value Example:

```
{
  "AssetType": "ouaccount",
  "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc",
  "UserId": "idcqa",
  "OrgId": "appdev",
  "TotalAccounts": 0,
  "TotalFtAccounts": 0,
  "AssociatedFtAccounts": [],
  "AssociatedNftAccount": ""
}
```

**`CreateTokenAccount`**

This method creates a fungible or non-fungible token account to associate with a user account.

A user can have multiple fungible token accounts with unique account IDs. Fungible token account IDs are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, the constant string `ft` separated by the tilde symbol (`~`), and a counter number that signifies the index of the fungible account that is being created separated by the tilde symbol (`~`).

A user can have only one non-fungible token account. Non-fungible token account IDs are unique and are formed by an SHA-256 hash of the `orgId` parameter, the `userId` parameter, and the constant string `nft` separated by the tilde symbol (`~`). All non-fungible tokens that a user owns, whether whole or fractional, are linked to this single non-fungible token account.

This method can be called only by a `Token Admin` of the chaincode.

```
Ctx.ERC1155Account.CreateTokenAccount(orgId string, userId string,
tokenType string) (ERC1155UserAccount, error)
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.
- `tokenType erc1155Token.TokenType` – The type of token account to create. The only supported token types are `nonfungible` and `fungible`.

Returns:

- On success, a JSON object of the token account that was created.

Return Value Example:

```
{
    "AssetType": "ouaccount",
    "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
```

```
37cc",
    "UserId": "idcqa",
    "OrgId": "appdev",
    "TotalAccounts": 1,
    "TotalFtAccounts": 1,
    "AssociatedFtAccounts": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "TokenId": ""
        }
    ],
    "AssociatedNftAccount": ""
}
```

**AssociateTokenToToken**

This method associates a user's fungible token account to a particular fungible token.

```
Ctx.ERC1155Account.AssociateTokenToToken(accountId string, tokenId string)
(interface{}, error)
```

Parameters:

- `accountId string` – The user account ID.
- `tokenId string` – The ID of the token.

Returns:

- On success, a JSON object of the user account, which shows that the fungible token was associated to the token account. For example, in the following example, the first object in the `AssociatedFtAccounts` array shows that the fungible token account ID and the token ID are associated.

Return Value Example:

```
{
    "AssetType": "ouaccount",
    "AccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
    "UserId": "idcqa",
    "OrgId": "appdev",
    "TotalAccounts": 1,
    "TotalFtAccounts": 1,
    "AssociatedFtAccounts": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "TokenId": "tokenOne"
        }
    ],
    "AssociatedNftAccount": ""
}
```

**GetAccountHistory**

This method returns history for a specified token account.

```
Ctx.ERC1155Account.GetAccountHistory(accountId string)
```

Parameters:

- `accountId string` – The token account ID.

Returns:

- On success, an array of JSON objects that describes the account history.

Return Value Example:


```
[
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-06T11:03:48Z",
        "TxId":
"c5180f3be3d9130f25a4b4e866f38a4283117dcbfbffb4f55e2c5b03dba0dd29",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "AssetType": "oaccount",
            "Balance": 100,
            "BapAccountVersion": 1
            "OrgId": "appdev",
            "TokenId": "loy1",
            "TokenName": "loyalty",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-06T11:02:39Z",
        "TxId":
"6f81b0c94b451d375a3892446aefbdf78d9fd1ac43699daa89f0dff10db5fd22",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e",
            "AssetType": "oaccount",
            "Balance": 0,
            "BapAccountVersion": 0
            "OrgId": "appdev",
            "TokenId": "loy1",
            "TokenName": "loyalty",
            "TokenType": "fungible",
```

```
            "UserId": "idcqa"
        }
    },
    {
        "IsDelete": "false",
        "Timestamp": "2023-06-05T16:28:46Z",
        "TxId":
"8185af648546e909488e72149be497b210f74f95ada252c42da9c35cb9d98691",
        "Value": {
            "AccountCategory": "",
            "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "AssetType": "oaccount",
            "Balance": 0,
            "BapAccountVersion": 0
            "OrgId": "appdev",
            "TokenId": "",
            "TokenName": "",
            "TokenType": "fungible",
            "UserId": "idcqa"
        }
    }
]
```

**GetAccountWithStatus**

This method returns token account details for a specified user, including account status. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC1155Account.GetAccountWithStatus(userAccountId string,
tokenId ...string) (interface{}, error)
```

Parameters:

- `userAccountId string` – The account ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details, including account status.

Return Value Example (Non-Fungible Token Account):

```
{
    "AccountId":
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
    "AssetType": "oaccount",
    "BapAccountVersion": 1,
    "NoOfNfts": 1,
    "OrgId": "appdev",
    "Status": "active",
```

```
    "TokenType": "nonfungible",
    "UserId": "idcqa"
}
```

Return Value Example (Fungible Token Account):

```
{
    "AccountCategory": "",
    "AccountId":
"oaccount~2de8db6b91964f8c9009136831126d3cfa94e1d00c4285c1ea3e6d1f36479
ed4",
    "AssetType": "oaccount",
    "Balance": 0,
    "BapAccountVersion": 0,
    "OrgId": "appdev",
    "Status": "active",
    "TokenId": "t1",
    "TokenName": "loyalty",
    "TokenType": "fungible",
    "UserId": "idcqa"
}
```

**`GetAccount`**

This method returns token account details for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

```
Ctx.ERC1155Account.Get(accountId string, tokenId ...string)
(interface{}, error)
```

Parameters:

- `userAccountId string` – The account ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON object that includes token account details.

Return Value Example (Non-Fungible Token Account):

```
{
    "AccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371",
    "BapAccountVersion": 0,
    "AssetType": "oaccount",
    "NoOfNfts": 4,
    "OrgId": "appdev",
    "TokenType": "nonfungible",
```

```
    "UserId": "idcqa"
}
```

Return Value Example (Fungible Token Account):

```
{
    "AssetType": "oaccount",
    "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
    "UserId": "idcqa",
    "OrgId": "appdev",
    "BapAccountVersion": 0,
    "TokenType": "fungible",
    "TokenId": "loy1",
    "TokenName": "loyalty",
    "Balance": 90,
    "AccountCategory": ""
}
```

**`GetAllAccounts`**

This method returns details of all user accounts.

```
Ctx.ERC1155Account.GetAllAccounts() (interface{}, error)
```

Parameters:

- none

Return Value Example:

```
[
    {
        "AssetType": "ouaccount",
        "AccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "UserId": "idcqa",
        "OrgId": "appdev",
        "TotalAccounts": 3,
        "TotalFtAccounts": 2,
        "AssociatedFtAccounts": [
            {
                "AccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
                "TokenId": "loy1"
            },
            {
                "AccountId":
"oaccount~58c5a6b09a41befca2a9ea2550439838c4dcf4d8a2a4f7c98e9319cf8479bfc4",
                "TokenId": ""
            }
        ],
        "AssociatedNftAccount":
```

```
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371"
    },
    {
        "AssetType": "ouaccount",
        "AccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352
003b",
        "UserId": "user1_minter",
        "OrgId": "appdev",
        "TotalAccounts": 2,
        "TotalFtAccounts": 1,
        "AssociatedFtAccounts": [
            {
                "AccountId":
"oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f6282379
74c",
                "TokenId": "loy1"
            }
        ],
        "AssociatedNftAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903
446"
    },
]
```

**GetAccountDetailsByUser**

This method returns an account summary for a specified user and details of fungible and non-fungible tokens that are associated with the user.

```
Ctx.ERC1155Account.GetAccountDetailsByUser(orgId string, userId
string) (interface{}, error)
```

Parameters:

- `orgId string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId string` – The user name or email ID of the user.

Returns:

- On success, a JSON account object that includes and account summary for the specified user and details of fungible and non-fungible tokens that are associated with the user. For fractional non-fungible tokens, the `tokenShare` property in the `associatedNFTs` section shows the share that the user owns.

Return Value Example:

```
{
    "AssociatedFTAccounts": [
        {
            "AccountId":
```

```
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "Balance": 90,
            "TokenId": "FT"
        },
    ],
    "AssociatedNFTAccount": {
        "AccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "AssociatedNFTs": [
            {
                "NFTTokenId": "FNFT",
                "TokenShare": 230
            },
            {
                "NFTTokenId": "NFT"
            },
            {
                "NFTTokenId": "NFT2"
            }
        ]
    },
    "UserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
}
```

**`GetUserByAccountId`**
This method returns the user details of a specified account ID.

```
Ctx.ERC1155Account.GetUserByAccountById(accountId string)
(map[string]interface{}, error)
```

Parameters:

- `accountId string` – The ID of the account.

Returns:

- On success, a JSON object of the user details (`orgId` and `userId`).

Return Value Example:

```
{
    "OrgId": "appdev",
    "UserId": "idcqa"
}
```

### Methods for Role Management

**AddRoleMember**

This method adds a role to a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.AddRoleMember(userRole string, userAccountId string,
asset interface{}) (interface{}, error)
```

Parameters:

- `userRole: string` – The name of the role to add to the specified user.
- `userAccountId: string` – The account ID of the user.
- `asset: interface{}` – The token asset.

Returns:

- On success, a message with account details.

Return Value Example:

```
{
    "msg": "Successfully added role minter to
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d2
2a (orgId : appdev, userId : idcqa)"
}
```

**IsInRole**

This method returns a Boolean value to indicate if a user has a specified role. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.IsInRole(userRole string, userAccountId string, asset
interface{}) (bool, error)
```

Parameters:

- `userRole: string` – The name of the role to search for.
- `userAccountId: string` – The account ID of the user.
- `asset: interface{}` – The token asset.

Return Value Example:

```
{
    "result": true
}
```

**RemoveRoleMember**

This method removes a role from a specified user and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Token.RemoveRoleMember(userRole string, userAccountId string,
asset interface{}) (interface{}, error)
```

Parameters:

- `userRole: string` – The name of the role to remove.
- `userAccountId: string` – The account ID of the user.
- `asset: interface{}` – The token asset.

Return Value Example:

```
{
    "msg": "Successfully removed role 'minter' from Account Id:
oaccount~ec7e4de2f81e3ea071710e07b6ff7d9346e84ef665ca4650885dbe8c3e2bd4c0
(Org-Id: appdev, User-Id: idcqa)"
}
```

**GetAccountsByRole**

This method returns a list of all account IDs for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Role.GetAccountsByRole(roleName string, token interface{})
(interface{}, error)
```

Parameters:

- `roleName: string` – The name of the role to search for.
- `token: interface{}` – The token asset.

Return Value Example:

```
{
    "accounts": [

"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",

"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b"
    ]
}
```

**GetUsersByRole**
This method returns a list of all users for a specified role and token. Fungible tokens are specified by the token ID. Non-fungible tokens are specified by the token name.

```
Ctx.ERC1155Role.GetUsersByRole(roleName string, token interface{})
(interface{}, error)
```

Parameters:

- `roleName: string` – The name of the role to search for.
- `token: interface{}` – The token asset.

Return Value Example:

```
{
    "Users": [
        {
            "AccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
            "OrgId": "appdev",
            "UserId": "idcqa"
        },
        {
            "AccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b",
            "OrgId": "appdev",
            "UserId": "user1"
        }
    ]
}
```

### Methods for Transaction History Management

**GetAccountTransactionHistory**
This method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner. For non-fungible tokens, this method can only be called when connected to the remote Oracle Blockchain Platform network.

```
Ctx.ERC1155Account.GetAccountTransactionHistory(tokenAccountId string)
(interface{}, error)
```

Parameters:

- `accountId string` – The user account ID.

Return Value Example:


```
[
    {
        "Balance": 90,
        "Timestamp": "2023-06-06T11:11:09Z",
        "TokenId": "FNFT",
        "TransactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "TransactedAmount": 10,
        "TransactionId":
"otransaction~0f4d96fbf8fed88ea8a313342897772109146 7c701848d595ebc3fffa88b365
7~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475b",
        "TransactionType": "DEBIT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
    },
    {
        "Timestamp": "2023-06-06T11:11:09Z",
        "TokenId": "NFT",
        "TransactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "TransactionId":
"otransaction~0f4d96fbf8fed88ea8a313342897772109146 7c701848d595ebc3fffa88b365
7~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc114",
        "TransactionType": "DEBIT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
    },
    {
        "Timestamp": "2023-06-06T11:06:54Z",
        "TokenId": "NFT",
        "TransactedAccount":
"oaccount~e88276a3be547e31b567346bddddde52d37734da4d5fae83ab2e5c98a10097371",
        "TransactionId":
"otransaction~6a13667ea3f6edc4c854e85b127526eccb58783f653c348b42a3869f0f29a4f
b~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "TransactionType": "MINT",
        "TriggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38"
    },
    {
        "Balance": 100,
        "Timestamp": "2023-06-05T16:34:33Z",
        "TokenId": "FNFT",
        "TransactedAccount":
"oaccount~e88276a3be547e31b567346bddddde52d37734da4d5fae83ab2e5c98a10097371",
        "TransactedAmount": 100,
        "TransactionId":
"otransaction~2bc15de1766d582d821bd8d61756bca02837dc683c0aa61f69657ccd1d95e33
5~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "TransactionType": "MINT",
        "TriggeredByUserAccountId":
```

```
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38"
    }
]
```

**GetTransactionById**

This method returns the transaction details for a specified transaction ID.

```
Ctx.ERC1155Transaction.GetTransactionById(trxId string) (interface{},
error)
```

Parameters:

- `trxId string` – The ID of the transaction.

Return Value Example:

```
{
    "history": [
        {
            "IsDelete": "false",
            "Timestamp": "2022-12-08T10:45:56Z",
            "TxId":
"2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948901aa98",
            "Value": {
                "Amount": 5,
                "AssetType": "otransaction",
                "FromAccountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc5
04b",
                "Timestamp": "2022-12-08T10:45:56Z",
                "ToAccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d
22a",
                "TokenId": "tokenOne",
                "TransactionId":
"otransaction~2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948
901aa98~9c3ce5f21abd98ca018c196086d73a812f2f49dba323f1de4f6867eecfeec8f
f",
                "TransactionType": "TRANSFER",
                "TriggeredByUserAccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec29
37cc"
            }
        }
    ],
    "transactionId":
"otransaction~2da02a53aa1032602df6c68c5628a4ab8b22ba107c0201520ce495948
901aa98~9c3ce5f21abd98ca018c196086d73a812f2f49dba323f1de4f6867eecfeec8f
f"
}
```

**DeleteHistoricalTransactions**
This method deletes transactions before a specified time stamp from the state database.

```
Ctx.ERC1155Transaction.DeleteHistoricalTransactions(referenceTime string)
(interface{}, error)
```

Parameters:

- `referenceTime string` – All transactions before this time stamp will be deleted.

Return Value Example:

```
{
    "Transactions": [

"otransaction~750f68538451847f57948f7d5261dcb81570cd9e429f928a4cb7bfa76392ecf
7"
    ],
    "msg": "Successfuly deleted transaction older than
date:2022-04-06T08:17:53Z"
}
```

### Methods for Token Behavior Management - Mintable Behavior

**MintBatch**
This method creates (mints) multiple tokens in a batch operation. This method creates only fungible tokens or fractional non-fungible tokens.
For fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. You cannot mint more than the `max_mint_quantity` property of the token, if that property was specified when the token was created or updated.
For non-fungible tokens, if the minter role is defined in the specification file, then any user with the minter role can call this method. If not, any user can use this method to mint tokens. Additionally, the caller must also be the creator of the token. There is no upper limit to the quantity of fractional non-fungible tokens that can be minted.
You cannot use this method to mint a whole non-fungible token.

```
Ctx.ERC1155Token.MintBatch(accountId string, tokens []interface{},
quantities []float64) (interface{}, error)
```

Parameters:

- `accountId string` – The account ID of the user.
- `tokenIds []string` – The list of token IDs to mint tokens for.
- `quantity []float64` – The list of quantities of tokens to mint, corresponding to the token ID array.

Returns:

- On success, a JSON object that includes details on the minted tokens.

Return Value Example:

```
{
    "details": [
        {
            "msg": "Successfully minted 100 tokens of fractional
tokenId: plot55 to Org-Id: appdev, User-Id: idcqa"
        },
        {
            "msg": "Successfully minted 100 tokens of tokenId:
'loyalty' to Token-Account-Id
'oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d4
10e'"
        }
    ],
    "msg": "Successfully minted batch of tokens for User-Account-Id
'ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c5
1f38' (Org-Id: 'appdev', User-Id: 'idcqa')"
}
```

### Methods for Token Behavior Management - Transferable Behavior

**`BatchTransferFrom`**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
This method does not validate that the caller of the method is the specified sender.

```
Ctx.ERC1155Token.BatchTransferFrom(fromAccountId string, toAccountId
string, tokens []interface{}, quantities []float64) (interface{},
error)
```

Parameters:

- `fromUserAccountId string` – The account ID of the sender and token owner in the current organization.
- `toUserAccountId string` – The account ID of the receiver.
- `tokenIds string[]` – A list of token IDs for the tokens to transfer.
- `quantity float64[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**SafeBatchTransferFrom**

This method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.
For NFTs, because the method transfers ownership of the NFT, the sender of the NFT must own the token.
For fractional NFTs, if a user (including the creator of the token) transfers all of the shares that they own, then they lose ownership of the token. If any share of a token is transferred to a user, that user automatically becomes one of the owners of the fractional NFT.
The caller of the method must be the specified sender.

```
Ctx.ERC1155Token.SafeBatchTransferFrom(fromAccountId string, toAccountId
string, tokens []interface{}, quantities []float64) (interface{}, error)
```

Parameters:

- `fromUserAccountId string` – The account ID of the sender and token owner in the current organization.
- `toUserAccountId string` – The account ID of the receiver.
- `tokenIds string[]` – A list of token IDs for the tokens to transfer.
- `quantity float64[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.

Returns:

- On success, a message with details for each token transfer.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10'
quantity from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a100973
71 (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f59034
46 (Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from
Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d41
0e (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f62823797
4c (Org-Id: appdev, User-Id: user1_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-
Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a100973
71 (Org-Id: appdev, User-Id: idcqa) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f59034
46 (Org-Id: appdev, User-Id: user1_minter)"
    }
]
```

**`BalanceOfBatch`**

This method completes a batch operation that gets the balance of token accounts. The account details are specified in three separate lists of organization IDs, user IDs, and token IDs. This method can be called only by a `Token Admin` of the chaincode or by account owners. Account owners can see balance details only for accounts that they own.

```
Ctx.ERC1155Account.BalanceOfBatch(accountIds []string, tokens
[]interface{}) (interface{}, error)
```

Parameters:

- `accountIds []string` – A list of the user account IDs.
- `tokenIds []string` – A list of the token IDs.

Return Value Example:

```
[
    {
        "OrgId": "appdev",
```

```
        "UserId": "idcqa",
        "UserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "TokenAccountId":
"oaccount~e88276a3be547e31b567346bddddde52d37734da4d5fae83ab2e5c98a10097371",
        "TokenId": "FNFT",
        "Balance": 100
    },
    {
        "OrgId": "appdev",
        "UserId": "idcqa",
        "UserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "TokenAccountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
        "TokenId": "FT",
        "Balance": 50
    },
    {
        "OrgId": "appdev",
        "UserId": "user1_minter",
        "UserAccountId":
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
        "TokenAccountId":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "TokenId": "FNFT",
        "Balance": 10
    }
]
```

**ExchangeToken**

This method exchanges tokens between specified accounts. This method only supports exchanging between an NFT (whole or fractional) and a fungible token or a fungible token and an NFT (whole or fractional). This method can be called only by the account owner.

```
Ctx.ERC1155Token.ExchangeToken(fromTokenId string, fromUserAccountId string,
fromTokenQuantity float64, toTokenId string, toUserAccountId string,
toTokenQuantity float64) (interface{}, error)
```

Parameters:

- `fromTokenId string` – The ID of the token that the sender owns.
- `fromUserAccountId string` – The account ID of the sender.
- `fromTokenQuantity float64` – The quantity of tokens from the sender to exchange with the receiver.
- `toTokenId string` – The ID of the token that the receiver owns.
- `toUserAccountId string` – The account ID of the receiver.
- `toTokenQuantity float64` – The quantity of tokens from the receiver to exchange with the sender.

Returns:

- On success, a message with token exchange details.

Return Value Example:

```
{
    "msg": "Succesfully exchanged 10 tokens of type 'nonfungible' with
tokenId: [r1] from Account
'oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097
371' (OrgId: appdev, UserId: idcqa) to 10 tokens of type 'fungible'
with tokenId: [loy1] from Account
'oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f6282379
74c' (OrgId: 'appdev', UserId: 'user1_minter')"
}
```

### Methods for Token Behavior Management - Burnable Behavior

**`Burn`**
This method deactivates, or burns, the specified fungible and non-fungible tokens.

```
Ctx.ERC1155Token.Burn(accountId string, tokens []interface{},
quantities []float64) (interface{}, error)
```

Parameters:

- `accountId string` – The account ID of the user.
- `tokenIds []string` – The list of token IDs to burn.
- `quantity []float64` – The list of quantities of tokens to burn, corresponding to the token ID array.

Returns:

- On success, a message with details about the burn operations.

Return Value Example:

```
[
    {
        "msg": "Successfully burned NFT token: 'art' from Account-Id:
oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282
c6 (Org-Id: appdev, User-Id: idcqa)"
    },
    {
        "msg": "Successfully burned 2 tokens of tokenId: FT from
Account-ID
oaccount~9a940587fd322ccc8400233244cd3b13f3aa2a52e418e4c71fb819a2217bc4
9e (Org-Id: AutoF1377358917, User-Id: idcqa)"
    },
    {
        "msg": "Successfully burned 2 token share of tokenId: FNFT
from Account-ID
```

```
oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a
(Org-Id: AutoF1377358917, User-Id: idcqa)"
    }
]
```

## Go Methods for ERC-1155 NFT Locking

Blockchain App Builder automatically generates methods that you can use to lock non-fungible tokens that use the extended ERC-1155 standard.

A locked token cannot be burned or transferred to other users. All other properties, such as the token's state, owner, and history are preserved. You can use the NFT locking functionality when transferring a token to another blockchain network, such as Ethereum or Polygon.

Before you can lock NFTs, you must assign the vault manager role to a user. The vault manager is a special type of role, a `TokenSys` role. `TokenSys` roles are different from asset-based roles such as burner, minter, and notary, and from administrative roles such as `Token Admin` and `Org Admin`. Currently Blockchain App Builder supports the `vault TokenSys` role. The single user who has the `vault` role for a chaincode is the vault manager of the chaincode, and can manage locked NFTs.

The typical flow for using the NFT locking functionality follows these steps.

- Create a non-fungible token that has the lockable behavior.
- Use the `AddTokenSysRole` method to give the `vault` role to a user, the vault manager.
- Call the `LockNFT` method to lock a non-fungible token, specified by the token ID.

### TokenSys Role Management Methods

**`AddTokenSysRole`**
This method adds a `TokenSys` role to a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenSysRole(orgId string, userId string, role
string) (interface{}, error) {
    userAccountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.AddTokenSysRole",
"TOKEN", map[string]string{"accountId": userAccountId})
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC1155Token.AddTokenSysRoleMember(role, userAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to give to the user.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully added role 'vault' to Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03e
ba (Org-Id: Org1MSP, User-Id: user1)"
}
```

**`IsInTokenSysRole`**

This method returns a Boolean value to indicate if a user has a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) IsInTokenSysRole(orgId string, userId string,
role string) (interface{}, error) {
    userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.IsInTokenSysRole",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC1155Token.IsInTokenSysRoleMember(role,
userAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to check.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "result": true,
    "msg": "Account Id
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1) has vault role"
}
```

**RemoveTokenSysRole**

This method removes a `TokenSys` role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) RemoveTokenSysRole(orgId string, userId string, role
string) (interface{}, error) {
    userAccountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
    if err != nil {
        return nil, err
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.RemoveTokenSysRole",
"TOKEN", map[string]string{"accountId": userAccountId})
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC1155Token.RemoveTokenSysRoleMember(role, userAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `role: string` – The name of the `TokenSys` role to remove.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully removed role 'vault' from Account Id:
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1)"
}
```

**TransferTokenSysRole**

This method transfers a `TokenSys` role from a user to another user. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) TransferTokenSysRole(role string, fromOrgId
string, fromUserId string, toOrgId string, toUserId string)
(interface{}, error) {
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.TransferTokenSysRole
", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
    }
    fromUserAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(fromOrgId, fromUserId,
constants.UserAccount)
    if err != nil {
        return nil, fmt.Errorf("error in TransferTokenSysRole. Error:
%s", err)
    }
    toUserAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(toOrgId, toUserId,
constants.UserAccount)
    if err != nil {
        return nil, fmt.Errorf("error in TransferTokenSysRole. Error:
%s", err)
    }
    return t.Ctx.ERC1155Token.TransferTokenSysRole(role,
fromUserAccountId, toUserAccountId)
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role from.
- `fromUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role from.
- `toOrgId: string` – The membership service provider (MSP) ID of the user to transfer the `TokenSys` role to.
- `toUserId: string` – The user name or email ID of the user to transfer the `TokenSys` role to.
- `role: string` – The name of the `TokenSys` role to transfer.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "msg": "Successfully transfered role 'vault' from Account Id:
ouaccount~f4e311528f03fffa7810753d643f66289ff6c9080fcf839902f28a1d3aff1789
(Org-Id: Org1MSP, User-Id: user1) to Account Id:
ouaccount~ae5be2ae8f98d6d32f5d02b43877d987114e7937c7bacbc30390dcce09996a19
(Org-Id: Org1MSP, User-Id: user2)"
}
```

**GetAccountsByTokenSysRole**

This method returns a list of all account IDs for a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetAccountsByTokenSysRole(role string) (interface{},
error) {
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Role.GetAccountsByTokenSysRole",
 "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC1155Token.GetAccountsByTokenSysRole(role)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
    "accountIds": [

"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba"
    ]
}
```

**GetUsersByTokenSysRole**

This method returns user information for all users with a specified `TokenSys` role. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) GetUsersByTokenSysRole(role string) (interface{},
error) {
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Role.GetUsersByTokenSysRole",
```

```
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    return t.Ctx.ERC1155Token.GetUsersByTokenSysRole(role)
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "Users":[
      {

"accountId":"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb8
3c874c2caf03eba",
         "orgId":"Org1MSP",
         "userId":"user1"
      }
   ]
}
```

### NFT Locking Methods

**`LockNFT`**

This method locks a specified non-fungible token. To lock a token, there must be a user with the `TokenSys vault` role, who acts as the vault manager. This method can be called only by the owner of the token.

```
func (t *Controller) LockNFT(orgId string, userId string, tokenId
string) (interface{}, error) {
    return t.Ctx.ERC1155Token.LockNFT(orgId, userId, tokenId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user (optional).
- `tokenID: string` – The ID of the token to lock.

Returns:

- On success, a JSON representation of the token object.

Return Value Example:


```
{
   "AssetType":"otoken",
   "Behaviors":[
      "indivisible",
      "mintable",
      "transferable",
      "burnable",
      "lockable",
      "roles"
   ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
   "CreationDate":"2023-10-20T12:22:47Z",
   "IsBurned":false,
   "IsLocked":true,
   "Mintable":{
      "Max_mint_quantity":20000
   },
   "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
   "Price":120,
   "Quantity":1,
   "Roles":{
      "minter_role_name":"minter"
   },
   "TokenDesc":"",
   "TokenId":"token1",
   "TokenMetadata":{
      "Description":"",
      "Image":"",
      "Painter_name":"",
      "Painting_name":""
   },
   "TokenName":"artcollection",
   "TokenStandard":"erc1155+",
   "TokenType":"nonfungible",
   "TokenUnit":"whole",
   "TokenUri":"token1.example.com"
}
```


**`IsNFTLocked`**

This method returns a Boolean value to indicate if a specified token is locked. This method can be called only by the token owner, the vault manager (the user with the `TokenSys vault` role), or a `Token Admin` of the chaincode.

```
func (t *Controller) IsNFTLocked(tokenId string) (interface{}, error) {
        auth, err :=
```

```
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.IsNFTLocked",
"TOKEN", map[string]string{"tokenId": tokenId})
        if err != nil && !auth {
            isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC1155Token.IsCallerTokenSysRoleHolder(constants.Vault)
            if error != nil {
                return nil, error
            }
            if !isCallerTokenSysRoleHolder {
                return nil, fmt.Errorf("error in authorizing the
caller  %s", err.Error())
            }
        }
        return t.Ctx.ERC1155Token.IsNFTLocked(tokenId)
    }
```

Parameters:

- `tokenID: string` – The ID of the token.

Returns:

- On success, a message that contains relevant details of the operation.

Return Value Example:

```
{
   "isNFTLocked":true
}
```

**`GetAllLockedNFTs`**

This method returns a list of all locked non-fungible tokens. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
func (t *Controller) GetAllLockedNFTs() (interface{}, error) {
        auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetAllLockedNFTs",
"TOKEN")
        if err != nil && !auth {
            isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC1155Token.IsCallerTokenSysRoleHolder(constants.Vault)
            if error != nil {
                return nil, error
            }
            if !isCallerTokenSysRoleHolder {
                return nil, fmt.Errorf("error in authorizing the
caller  %s", err.Error())
            }
        }
        return t.Ctx.ERC1155Token.GetAllLockedNFTs()
    }
```

Parameters:

- None

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:


```
[
   {
      "key":"token1",
      "valueJson":{
         "AssetType":"otoken",
         "Behaviors":[
            "indivisible",
            "mintable",
            "transferable",
            "burnable",
            "lockable",
            "roles"
         ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "CreationDate":"2023-10-20T12:22:47Z",
         "IsBurned":false,
         "IsLocked":true,
         "Mintable":{
            "Max_mint_quantity":20000
         },
         "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
         "Price":120,
         "Quantity":1,
         "Roles":{
            "minter_role_name":"minter"
         },
         "TokenDesc":"",
         "TokenId":"token1",
         "TokenMetadata":{
            "Description":"",
            "Image":"",
            "Painter_name":"",
            "Painting_name":""
         },
         "TokenName":"artcollection",
         "TokenStandard":"erc1155+",
         "TokenType":"nonfungible",
         "TokenUnit":"whole",
         "TokenUri":"token1.example.com"
```

```
      }
   }
]
```

**GetAllLockedNFTsByOrg**

This method returns a list of all locked non-fungible tokens for a specified organization and optionally a specified user. This method can be called only by the vault manager (the user with the `TokenSys vault` role) or a `Token Admin` of the chaincode.

```
func (t *Controller) GetLockedNFTsByOrg(orgId string,
userId ...string) (interface{}, error) {
        auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155Token.GetLockedNFTsByOrg",
 "TOKEN")
        if err != nil && !auth {
            isCallerTokenSysRoleHolder, error :=
t.Ctx.ERC1155Token.IsCallerTokenSysRoleHolder(constants.Vault)
            if error != nil {
                return nil, error
            }
            if !isCallerTokenSysRoleHolder {
                return nil, fmt.Errorf("error in authorizing the
caller  %s", err.Error())
            }
        }
        return t.Ctx.ERC1155Token.GetLockedNFTsByOrg(orgId, userId...)
    }
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user (optional).

Returns:

- On success, an array of the locked non-fungible token objects.

Return Value Example:

```
[
   {
      "key":"token1",
      "valueJson":{
         "AssetType":"otoken",
         "Behaviors":[
            "indivisible",
            "mintable",
            "transferable",
            "burnable",
            "lockable",
            "roles"
```

```
         ],

"CreatedBy":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b
6a7733463",
         "CreationDate":"2023-10-20T12:22:47Z",
         "IsBurned":false,
         "IsLocked":true,
         "Mintable":{
            "Max_mint_quantity":20000
         },
         "On_sale_flag":false,

"Owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a77
33463",
         "Price":120,
         "Quantity":1,
         "Roles":{
            "minter_role_name":"minter"
         },
         "TokenDesc":"",
         "TokenId":"token1",
         "TokenMetadata":{
            "Description":"",
            "Image":"",
            "Painter_name":"",
            "Painting_name":""
         },
         "TokenName":"artcollection",
         "TokenStandard":"erc1155+",
         "TokenType":"nonfungible",
         "TokenUnit":"whole",
         "TokenUri":"token1.example.com"
      }
   }
]
```


## Go Methods for ERC-1155 Token Account Status

Blockchain App Builder automatically generates methods that you can use to manage account status for tokens that use the extended ERC-1155 standard.

You can use the following methods to put token user accounts in the active, suspended, or deleted states.

When an account is suspended, the account user cannot complete any write operations, which include minting, burning, and transferring tokens. Additionally, other users cannot transfer tokens to a suspended account. A suspended account can still complete read operations.

An account with a non-zero token balance cannot be deleted. You must transfer or burn all tokens in an account before you can delete the account. After an account is in the deleted state, the account state cannot be changed back to active or suspended.

- Automatically Generated Account Status Methods
- Account Status SDK Methods

### Automatically Generated Account Status Methods

**`GetAccountStatus`**

This method gets the current status of the token account. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
func (t *Controller) GetAccountStatus(orgId string, userId string,
tokenId ...string) (interface{}, error) {
      userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
      }
      auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155AccountStatus.Get",
"TOKEN", map[string]string{"accountId": userAccountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller
%s", err.Error())
      }
      tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId,
tokenId...)
      if err != nil {
            return nil, fmt.Errorf("error in GetAccountStatus. Error:
%s", err)
      }
      tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
      accountStatus, err :=
t.Ctx.ERC1155AccountStatus.GetAccountStatus(tokenAccountId)
      if err != nil {
            return
t.Ctx.ERC1155AccountStatus.GetDefaultAccountStatus(tokenAccountId)
      }
      return accountStatus, nil
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the

account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**

This method gets the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

```
func (t *Controller) GetAccountStatusHistory(orgId string, userId string,
tokenId ...string) (interface{}, error) {
      userAccountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
      if err != nil {
            return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
      }
      auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155AccountStatus.Get", "TOKEN",
map[string]string{"accountId": userAccountId})
      if err != nil && !auth {
            return nil, fmt.Errorf("error in authorizing the caller %s",
err.Error())
      }
      tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId,
tokenId...)
      if err != nil {
            return nil, fmt.Errorf("error in GetAccountStatusHistory. Error:
%s", err)
      }
      tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
      statusId, err :=
t.Ctx.ERC1155AccountStatus.GenerateAccountStatusId(tokenAccountId)
      if err != nil {
            return nil, err
      }
      accountStatusHistory, err :=
t.Ctx.ERC1155AccountStatus.History(statusId)
      if err != nil {
            return []map[string]interface{}{}, nil
      }
```

```
      return accountStatusHistory, nil
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, the account status history in JSON format.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
  },
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
  }
]
```

**`ActivateAccount`**

This method activates a token account. This method can be called only by a `Token Admin` of the chaincode. Deleted accounts cannot be activated. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
func (t *Controller) ActivateAccount(orgId string, userId string,
tokenId ...string) (interface{}, error) {
    userAccountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating accountId of
(Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155AccountStatus.ActivateAccount",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId, tokenId...)
    if err != nil {
        return nil, fmt.Errorf("error in GetAccountTransactionHistory.
Error: %s", err)
    }
    tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
    return t.Ctx.ERC1155Account.ActivateAccount(tokenAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
```

```
    "Status": "active"
}
```

**SuspendAccount**

This method suspends a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is suspended, you cannot complete any operations that update the account. A deleted account cannot be suspended.

```
func (t *Controller) SuspendAccount(orgId string, userId string,
tokenId ...string) (interface{}, error) {
    userAccountId, err :=
t.Ctx.ERC1155Account.GenerateAccountId(orgId, userId,
constants.UserAccount)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating
accountId of (Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155AccountStatus.SuspendAccou
nt", "TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId,
tokenId...)
    if err != nil {
        return nil, fmt.Errorf("error in GetAccountTransactionHistory.
Error: %s", err)
    }
    tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
    return t.Ctx.ERC1155Account.SuspendAccount(tokenAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
```

```
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "suspended"
}
```

**DeleteAccount**

This method deletes a token account. This method can be called only by a `Token Admin` of the chaincode. After an account is deleted, you cannot complete any operations that update the account. The deleted account is in a final state and cannot be changed to any other state. To delete an account, the account balance must be zero.

```
func (t *Controller) DeleteAccount(orgId string, userId string,
tokenId ...string) (interface{}, error) {
    userAccountId, err := t.Ctx.ERC1155Account.GenerateAccountId(orgId,
userId, constants.UserAccount)
    if err != nil {
        return nil, fmt.Errorf("error in getting the generating accountId of
(Org-Id: %s, User-Id: %s)", orgId, userId)
    }
    auth, err :=
t.Ctx.ERC1155Auth.CheckAuthorization("ERC1155AccountStatus.DeleteAccount",
"TOKEN")
    if err != nil && !auth {
        return nil, fmt.Errorf("error in authorizing the caller  %s",
err.Error())
    }
    tokenAccount, err := t.Ctx.ERC1155Account.Get(userAccountId, tokenId...)
    if err != nil {
        return nil, fmt.Errorf("error in GetAccountTransactionHistory.
Error: %s", err)
    }
    tokenAccountId, err := util.GetAccountProperty(tokenAccount,
constants.AccountId)
    return t.Ctx.ERC1155Account.DeleteAccount(tokenAccountId)
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user in the current organization.
- `userId: string` – The user name or email ID of the user.
- `tokenId ...string` – For a non-fungible token account, an empty string. For a fungible token account, the token ID.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "deleted"
}
```

### Account Status SDK Methods

**`GetDefaultAccountStatus`**
This method gets the current status of a token account, with the status as `active` for any account that does not have account status stored in the ledger (because the account was created prior to the account status functionality).

```
Ctx.GetDefaultAccountStatus(accountId string) (ERC1155AccountStatus,
error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
      "AssetType": "oaccountStatus",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "Status": "active"
}
```

**`GetAccountStatus`**
This method gets the current status of the token account.

```
Ctx.ERC1155AccountStatus.GetAccountStatus(accountId string)
(NFTAccountStatus, error)
```

Parameters:

- `accountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the token account status. If no status is found in the ledger for the account because the account was created before the account status functionality was available, the status is listed as `active` in the response.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "active"
}
```

**GetAccountStatusHistory**
This method gets the history of the account status.

```
Ctx.ERC1155AccountStatus.History(statusId string) (interface{}, error)
```

Parameters:

- `statusId: string` – The ID of the account status object.

Returns:

- On success, a JSON representation of the account status history.

Return Value Example:

```
[
  {
    "IsDelete": "false",
    "Timestamp": "2022-12-02T16:20:34+05:30",
    "TxId":
"af1601c7a14b4becf4bb3b285d85553b39bf234caaf1cd488a284a31a2d9df78",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "AssetType": "oaccountStatus",
      "Status": "suspended",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7"
    }
  },
  {
    "IsDelete": "false",
```

```
    "Timestamp": "2022-12-02T16:19:15+05:30",
    "TxId":
"4b307b989063e43add5357ab110e19174d586b9746cc8a30c9aa3a2b0b48a34e",
    "Value": {
      "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
      "AssetType": "oaccountStatus",
      "Status": "active",
      "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7"
    }
  }
]
```

**`ActivateAccount`**

This method activates a token account. For any accounts created prior to the account status functionality being available, you must call this method to set the account status as `active`.

```
Ctx.ERC1155Account.ActivateAccount(tokenAccountId string)
(interface{}, error)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f
79d5e96d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f
9c1",
    "Status": "active"
}
```

**`SuspendAccount`**

This method suspends a token account.

```
Ctx.ERC1155Account.SuspendAccount(tokenAccountId string) (interface{},
error)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "suspended"
}
```

**`DeleteAccount`**
This method deletes a token account.

```
Ctx.ERC1155Account.DeleteAccount(tokenAccountId string) (interface{}, error)
```

Parameters:

- `tokenAccountId: string` – The ID of the token account.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "AssetType": "oaccountStatus",
    "StatusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e9
6d7",
    "AccountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "Status": "deleted"
}
```

# Deploying and Testing Token Chaincode

You deploy token chaincode projects by following the same steps as other chaincode projects, but there are some special issues to consider.

You can deploy chaincode to only one organization or instance when you use the standard deployment steps with Blockchain App Builder. When a token chaincode is deployed, the list

of `Token Admin` users is specified. The `Token Admin` user can add or remove other users by calling the `addAdmin` and `removeAdmin` methods.

To deploy to multiple organizations or instances, package the chaincode and then manually deploy it to all the instances. For more information, see the applicable following topic:

- CLI: Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network
- Visual Studio Code: Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network

To test token projects locally with multiple users, see the applicable following topic:

- CLI: Testing Multiple Token Users Locally
- Visual Studio Code: Testing Multiple Token Users Locally

To test token projects on a remote Oracle Blockchain Platform network, see the applicable following topic:

- CLI: Testing Token Projects on a Remote Oracle Blockchain Platform Network
- Visual Studio Code: Testing Token Projects on a Remote Oracle Blockchain Platform Network

#### Adding Enrollments for Token Use Cases

Oracle Blockchain Platform supports enrollments to the REST proxy. You use enrollments with token chaincodes to ensure the identities of the users completing token transactions. To do this, when you add enrollments for token use cases, specify a user ID for each enrollment, and specify one and only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to the REST Proxy.

## Working With the Sample Token Specification Files

You can use the sample token specification files that come with Blockchain App Builder to investigate the complete life cycle of a token.

To get the Blockchain App Builder samples, in the service console open the **Developer Tools** tab and then select the **Blockchain App Builder** pane. Click **Download Specification Samples and Related Code** and then extract the downloaded archive file (`obp-app-builder-samples.zip`). The archive file includes token specification files for the Fiat Money Token, Loyalty Token, NFT Art Collection Marketplace, and Fractional NFT in Real Estate samples.

For the Fiat Money Token and Loyalty Token samples, the archive file also includes scaffolded token chaincode in Go and a corresponding Postman collection file (specific to Go), which you can use to test token life cycle operations on an Oracle Blockchain Platform network.

The NFT Art Collection Marketplace and Fractional NFT in Real Estate samples includes specification files that are designed for development in TypeScript. The NFT Art Collection Marketplace sample includes scaffolded token chaincode in Typescript and a corresponding Postman collection file (specific to Typescript), which you can use to test token life cycle operations for a non-fungible token on an Oracle Blockchain Platform network. There are two NFT Art Collection Marketplace samples: one for the ERC-721 standard and one for the ERC-1155 standard.

The token asset name is specified in the sample specification file. A token is defined by behaviors, which are also in the specification file. The sample token assets also include custom properties.

For more information about using the token samples, see the `readme.md` file in the sample archive file.

For more information about creating specification files in Blockchain App Builder, see the following topics:

- Input Specification File
- Input Specification File for Token Taxonomy Framework
- Input Specification File for ERC-721
- Input Specification File for ERC-1155

For more information about deploying and testing chaincode projects, see Deploying and Testing Token Chaincode.

## Disaster Recovery Support for Tokenization

You can configure Oracle Blockchain Platform for disaster recovery support in tokenization scenarios.

Before you can set up disaster recovery support in a tokenization scenario, the blockchain network must be configured for disaster recovery, using the following guidelines. For more information, see Add Oracle Blockchain Platform Participant Organizations to the Network.

- In addition to the founder instance of Oracle Blockchain Platform, create at least two more participant instances and join them to the blockchain network.
- Add at least two orderers from each participant instance to the blockchain network. Typically you add at least three orderers from each participant instance.
- Join the participant organizations to the same channels and install chaincodes on the participant instances.
- Ensure that the orderers from the participant instances are joined to the channels on the founder instance.

You can then prepare for disaster recovery in a tokenization scenario by designating a second instance of Oracle Blockchain Platform as a disaster recovery organization. When a primary Oracle Blockchain Platform instance is down and unable to send transactions to the chaincode, a secondary Oracle Blockchain Platform instance set up as a disaster recovery organization can be used to send the transaction to the chaincode on behalf of the primary instance.

In the following example, the founder organization on the primary instance is Org1MSP and the participant organization on the disaster recovery instance is Org2MSP.

To set up a second instance for disaster recovery in tokenization scenarios, complete the following steps.

1. Create custom enrollments on the disaster recovery organization and add a custom attribute for `primaryOrgMSPId` that is the ID of the founder organization (Org1MSP). You can use a tool such as Postman to create the custom enrollment. This enrollment indicates that Org2MSP is a disaster recovery organization for the primary organization, Org1MSP.

The following example shows an example request body for the REST endpoint `{{bc-url}}/console/admin/api/v2/nodes/restproxies/{{bc-restproxy-id}}/enrollments`. You can get the `bc-restproxy-id` value in the response from a GET request for the REST endpoint `{{bc-url}}/console/admin/api/v2/nodes`.

```
{
   "enrollmentId": "<enrollmentId>",
    "attributes":{
        "primaryOrgMSPId": "<primaryOrgId>"
    }
 }
```

The following example is the expected response.

```
{
"respMesg": "SUCCESS"
}
```

2. Add users to the custom enrollment. You can use Postman or the Oracle Blockchain Platform console to add users. The enrollment IDs and user names in the disaster recovery organization (Org2MSP) must match the enrollment IDs and user names in the primary organization (Org1MSP). The only difference is the custom attribute, `primaryOrgMSPId`, which points to the primary organization (Org1MSP).
   The following example shows an example request body for the REST endpoint `{{bc-url}}/console/admin/api/v2/nodes/restproxies/{{bc-restproxy-id}}/enrollments/{{bc-enrolment-id}}/users .`

```
{
    "userName": "<userId>"
 }
```

The following example is the expected response.

```
{
"respMesg": "SUCCESS"
}
```

8

# Deploy and Manage Chaincodes

To learn more about deploying, monitoring, and upgrading chaincodes, select the section based on the platform version of Hyperledger Fabric that your instance is running on.

- Deploy and Manage Chaincodes on Hyperledger Fabric v2.x
- Deploy and Manage Chaincodes on Hyperledger Fabric v1.4.7

## Deploy and Manage Chaincodes on Hyperledger Fabric v2.x

This topic contains information to help you deploy, monitor, and find information about the chaincodes on the network.

**Topics**



### Typical Workflow to Deploy Chaincodes

(Hyperledger Fabric v2.x) Here are the common tasks for deploying chaincodes.

You must be an administrator to complete these tasks.

| Task | Description | More Information |
|---|---|---|
| Use the wizard to fully or partially deploy a chaincode | For testing, use Quick Deployment to perform the deployment in one step, using default settings.<br>For production, use Advanced Deployment to specify the deployment settings such as which peers to install the chaincode on and the endorsement policy you want to use. | Use Quick Deployment<br>Use Advanced Deployment |
| Deploy a chaincode | Deploying a chaincode consists of approving and committing the chaincode definition. | Deploy a Chaincode<br>Chaincode Life Cycle |
| Upgrade a chaincode | Upload a newer version of a chaincode package, or update a chaincode definition. | Upgrade a Chaincode |

# Use Quick Deployment

(Hyperledger Fabric v2.x) Use the quick deployment option to complete a one-step chaincode deployment. This option is recommended for chaincode testing.

The quick deployment uses default settings, installs the chaincode on all peers in the channel, deploys the chaincode using the default endorsement policy, and enables the chaincode in the REST proxy.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x).
- You can use the advanced deployment option to put your chaincode into production on the network. See Use Advanced Deployment.

You must be an administrator to complete this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Quick Deployment**.

   The Deploy Chaincode (Quick) page is displayed.
4. In the **Package Label** field, enter a description of the chaincode package.

   Use the following guidelines when labeling the chaincode:

   - Use ASCII alphanumeric characters, dashes (-), and underscores (_).
   - The label must start and end only with ASCII alphanumeric characters. For example, you can't use labels such as `_mychaincode` or `mychaincode_`.

- Dashes (-) and underscores (_) must be followed by ASCII alphanumeric characters. For example, you can't use names like `my--chaincode` or `my-_chaincode`.
- The package label can be up to 50 characters long.

5. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field enter a string value to specify the chaincode's version number.

   Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters.
   - The name and version can each be up to 64 characters long.
   - The chaincode version can also contain periods (.) and plus signs (+).

6. If the chaincode requires initialization, select **Init-required**.

   If **Init-required** is selected, the client application must invoke the `Init` function explicitly, by specifying the `isInit` flag, before calling any other function.

7. Review the other default settings and modify them as needed.
8. If you are deploying chaincode source in a `.zip` file, leave **Is Packaged Chaincode** deselected. If you are deploying a chaincode package in a `.tar.gz` file, select **Is Packaged Chaincode**.
9. Click **Upload Chaincode File** and browse for the chaincode file to upload and deploy.
10. Click **Submit**.

    The chaincode is installed on the channel's peers and deployed.

On the Channels tab, click the name of the channel that you deployed the chaincode to, and then click **Deployed Chaincodes**. The deployed chaincode's name, version, sequence number, and package ID are displayed in the summary table, as well as the approved and committed statuses.

## Use Advanced Deployment

(Hyperledger Fabric v2.x) Use the advanced deployment option to specify the parameters required to deploy a chaincode into a production environment. For example, you'll specify which peers to install the chaincode on and the endorsement policy to use.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x).
- You can use the quick deployment option for chaincode testing. Quick deployment is a one-step deployment that uses default settings, installs the chaincode on all peers in the channel, and deploys the chaincode using a default endorsement policy. See Use Quick Deployment.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.

3. Click **Advanced Deployment**.

   The Deploy Chaincode (Advanced) Step 1 of 2: Install page is displayed.

4. In the **Package Label** field, enter a description of the chaincode package.

   Use the following guidelines when labeling the chaincode:

   - Use ASCII alphanumeric characters, dashes (-), and underscores (_).
   - The label must start and end only with ASCII alphanumeric characters. For example, you can't use labels such as `_mychaincode` or `mychaincode_`.
   - Dashes (-) and underscores (_) must be followed by ASCII alphanumeric characters. For example, you can't use names like `my--chaincode` or `my-_chaincode`.
   - The package label can be up to 50 characters long.

5. Select the languange that the chaincode is written in, and select one or more network peers to install the chaincode onto. To provide high availability, choose the appropriate number of peers from each partition. The peers you choose must be joined to the channel that you'll instantiate the chaincode on.

6. If you are deploying chaincode source in a `.zip` file, leave **Is Packaged Chaincode** deselected. If you are deploying a chaincode package in a `.tar.gz` file, select **Is Packaged Chaincode**.

7. Click **Upload Chaincode File** and browse for the chaincode file to upload and deploy. Click **Next**.

   The chaincode is installed and the Deploy Chaincode (Advanced) Step 2 of 2: Deploy page is displayed.

8. Decide if you want to deploy the chaincode now or later.

   - Click **Close** to close the wizard and deploy later.
   - To deploy now, select the channel to deploy the chaincode on.

9. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field enter a string value to specify the chaincode's version number.

   Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters.
   - The name and version can each be up to 64 characters long.
   - The chaincode version can also contain periods (.) and plus signs (+).

10. If the chaincode requires initialization, select **Init-required**.

    If **Init-required** is selected, the client application must invoke the `Init` function explicitly, by specifying the `isInit` flag, before calling any other function.

11. If required, enter an endorsement policy and private data collections, and then click **Next**. For more information about endorsement policies, see Specify an Endorsement Policy. For more information about private data collections, see Add Private Data Collections.

    Note the following information:

- Deployment approves, commits, and initializes the chaincode on the channel.
- If you do not change the endorsement policy, Oracle Blockchain Platform uses the default endorsement policy. The default endorsement policy is defined in the /Channel/Application/Endorsement policy of the channel where you are deploying the chaincode. The default endorsement policy gets an endorsement from any peer from any organization on the network.
- When deployment is complete, the peers are able to accept chaincode invocations and can endorse transactions.

The chaincode is deployed.

# Deploy a Chaincode

(Hyperledger Fabric v2.x) To deploy a chaincode, it must be approved by organizations and then committed to a channel. After a chaincode is deployed, peers are able to accept chaincode invocations and can endorse transactions.

Note the following information:

- You must install the chaincode on the required peers before you can deploy it.
- You can deploy more than one chaincode on a channel.
- The process to deploy the sample chaincodes is different than the deployment process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x).

You must be an administrator to complete this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, locate the chaincode package and click its **More Actions** menu, and select **Deploy**.

   The Deploy Chaincode dialog is displayed.
3. Enter information about where and how to deploy the chaincode.

| Field | Description |
|---|---|
| Channel | Select the channel for the chaincode to run on. |
| Chaincode Name | Enter a unique name, up to 64 characters long, for the deployed chaincode.<br>• Use ASCII alphanumeric characters, dashes (-), and underscores (_).<br>• The name must start and end only with ASCII alphanumeric characters.<br>• Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. |
| Version | Enter a string value, up to 64 characters long, to specify the chaincode's version number.<br>• Use ASCII alphanumeric characters, dashes (-), underscores (_), periods (.) and plus signs (+). |
| Init-required | Select if the chaincode requires initialization. If selected, the client application must invoke the `Init` function explicitly, by specifying the `isInit` flag, before calling any other function. |
| Endorsement Policy | In this section, specify the policy required to endorse the chaincode.<br>If you don't specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network. |

| Field | Description |
|---|---|
| Private Data Collection | In this section, add one or more private data collections. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel you deploy the chaincode on. |

4. Click **Deploy**.

   The chaincode is deployed.

5. To confirm that the chaincode was deployed, go to the Channels tab and click the name of the channel that you deployed the chaincode on. Go to the Deployed Chaincodes tab and confirm that the chaincode is listed in the summary table.

# Chaincode Life Cycle

(Hyperledger Fabric v2.x) The chaincode life cycle describes the process of installing chaincode on peers and deploying it on a channel.

The chaincode life cycle is based on the capabilities of the Hyperledger Fabric v2.x platform, which allows for the decentralized governance of chaincodes. Multiple organizations can agree on chaincode parameters, including the chaincode endorsement policy, before a chaincode can interact with the ledger. These functions are implemented in the new quick deployment and advanced deployment options, as well as in the REST API. For more information about the new life cycle, see Fabric chaincode lifecycle in the Hyperledger Fabric v2.2.4 documentation.

Typically, to deploy an installed chaincode, you use quick deployment or advanced deployment in the console. The deployment process includes packaging and installing the chaincode as well as approving and committing the chaincode definition. You can also use the REST API to complete the approval and commitment operations separately.

### Package and Install a Chaincode

When you install chaincode in Oracle Blockchain Platform, the chaincode is packaged, installed, and a package ID is generated automatically. The package ID is displayed on the **Chaincodes** tab of the console.

### Approve a Chaincode Definition

Before a chaincode can be deployed to a channel, the chaincode definition must be approved by enough organizations to satisfy the LifecycleEndorsement policy of the channel. The default LifecycleEndorsement policy in Oracle Blockchain Platform lets any organization approve the chaincode definition (as opposed to a majority of organizations). The chaincode definition includes the following parameters, which must be the same for all organizations: `Chaincode Name`, `Version`, `Sequence`, `Endorsement Policy`, `Private Data Collection`, and `Init-required`. A chaincode definition can also include a `Package ID`, which does not have to be the same for all organizations.

After a chaincode definition is approved, one organization can collect endorsements from peers of the approving organizations and then commit the chaincode definition to the channel.

To approve a chaincode definition by using the REST API, see Approve a Chaincode Definition in a Channel.

In the console, when you use quick deployment or advanced deployment the approval and commitment steps are both attempted.

### Commit a Chaincode Definition

To commit an approved chaincode definition by using the REST API, see Commit a Chaincode Definition in a Channel.

In the console, you can see chaincode definitions that are approved but not committed on the Deployed Chaincodes page for the channel. You can use the **More Actions** menu to commit the approved chaincode.

### Chaincode Life Cycle Scenarios

| Scenario | Description |
|---|---|
| Join a channel | Typically in the console you do not approve a chaincode definition without then committing it. If you join a shared channel where a chaincode definition was committed by another organization, you will see the chaincode definition listed as committed but not approved on the Deployed Chaincodes page for the channel. You can use the **More Actions** menu to approve the chaincode definition and also to associate a package ID. You do not need to commit the package definition again. |
| Update an endorsement policy | You can update the endorsement policy in the chaincode definition without reinstalling the chaincode. On the Deployed Chaincodes page for the channel, use the More Actions menu to upgrade the chaincode definition. Expand **Endorsement Policy** and specify a new policy, then click **Upgrade**. |
| Approve a definition without installing | In a multiple organization scenario, to approve a chaincode definition without installing the chaincode package, do not specify a package ID. You endorse the definition of the chaincode that is committed to the channel, but the chaincode is not installed on peers in your organization. You will not be able to use the chaincode to endorse transactions or query the ledger. |
| Disagreement on definitions | In a multiple organization scenario, an organization that doesn't approve a chaincode definition or approves a different chaincode definition is not able to run the chaincode on their peers. If other organizations get enough endorsements to commit the definition to the channel, those organizations can use the chaincode. Transactions are still added to the ledger on the peers of all organizations.<br>If organizations do not agree on a chaincode definition and no organizations get enough endorsements to commit the definition to the channel, the definition cannot be committed and therefore the chaincode cannot run. |

| Scenario | Description |
|---|---|
| Multiple organizations install different packages | You can specify a different package ID when you approve a chaincode definition for a channel with multiple organizations. If the definition name and endorsement policy are the same, then channel members can install chaincode that is specific to their organization, but which reads and writes data to the same chaincode namespace. |
| Create multiple chaincodes from one package | Similarly, you can approve and commit the same chaincode package multiple times, specifying a different name for each definition. Multiple instances of the chaincode run on the channel. If you also specify a different endorsement policy for each definition, then each chaincode instance is subject to a different endorsement policy. |

# Specify an Endorsement Policy

(Hyperledger Fabric v2.x) You can add an endorsement policy when you deploy a chaincode. An endorsement policy specifies the members with peers that must approve, or properly endorse, a chaincode transaction before it's added to a block and submitted to the ledger.

Endorsement guarantees the legitimacy of a transaction. When you deploy a chaincode on a channel, you can specify an endorsement policy. If you don't specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

A member's endorsing peers must have ReaderWriter permissions on the channel. When a transaction is processed, each endorsing peer returns a signed read-write set. After the client has enough endorsements to meet the endorsement policy requirements, then the client bundles the common read-write set with the signature from the endorsing peers and sends everything to the ordering service, which orders and commits the transactions into blocks and then to the ledger.

You can go to the Channels tab to view a deployed chaincode's endorsement policy. See View an Endorsement Policy. You can't modify a deployed chaincode's endorsement policy. If you need to change an endorsement policy, then you must redeploy the chaincode or upgrade it to another version and specify a different endorsement policy.

You must be an administrator to complete this task.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode package that you want to deploy and use the **More Actions** menu to begin the deployment process.
3. On the Deploy Chaincode window, expand **Endorsement Policy**.
4. Select **Default**, **Signature Policy** or **Channel Config Policy**, and then specify an expression for the endorsement policy.

   For more information about endorsement policies, see Endorsement policies in the Hyperledger Fabric documentation.
5. Complete the other fields on the Deploy Chaincode page as needed.
6. Click **Deploy**.

# View an Endorsement Policy

(Hyperledger Fabric v2.x) You can view a deployed chaincode's endorsement policy.

You might need to view a deployed chaincode's endorsement policy to see how it was set up, how you need to choose transaction endorsers based on the policy, or to help resolve an endorsement failure.

You can't modify the endorsement policy for a deployed chaincode. If you need to change an endorsement policy, then you must redeploy the chaincode or upgrade it to another version and specify a different endorsement policy.

1. Go to the console and select the Channels tab.
2. Click the name of the channel where the chaincode is deployed, and then click **Deployed Chaincodes**.
3. In the table, click the **More Actions** menu icon for the chaincode, and then click **View Chaincode Definition**.

   The Chaincode Definition window is displayed.
4. Expand **Endorsement Policy**.

   The chaincode endorsement policy is displayed.

# Find Information About Chaincodes

(Hyperledger Fabric v2.x) You can find information about the chaincodes in your network, including how many peers the chaincode is installed on and if the chaincode has been deployed. You can view information about chaincode packages and chaincode definitions.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes page is displayed and the chaincode table lists the chaincode packages that are available on the network, with information about how many peers a package is installed on and how many channels a package is deployed on.
2. In the table, click a chaincode package to see more information about which peers it's installed on, and its names and versions for the channels it's deployed on.
   - When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Chaincodes tab.
   - If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. To list the chaincode, start at least one peer node that has the chaincode installed on it.
   - Use the **More Actions** menu icon to deploy the chaincode package to a different channel, or to the same channel but with a different chaincode definition. You can also download the chaincode package and delete the chaincode package. You might delete a chaincode package to free up space for installing other chaincodes. When you delete a chaincode package, it is not recoverable.
3. To see the definitions of deployed chaincodes, select the Channels tab.
4. Click the name of the channel where the chaincode is deployed, and then click **Deployed Chaincodes**.

   In the table, you can use the **More Actions** menu to get information about a chaincode definition or to upgrade a chaincode.

## Delete a Chaincode

You can delete obsolete or unused chaincode packages to free up disk space.

You might delete a chaincode package to free up space for installing other chaincodes. When you delete a chaincode package, it is not recoverable.

1. Go to the console and select the **Chaincodes** tab.

   The Chaincodes page is displayed. The chaincode table lists the chaincode packages that are available on the network, with information about how many peers a package is installed on and how many channels a package is deployed on.

2. In the table, click the **More Actions** menu item for the chaincode to delete, and then click **Delete**.
3. Click **Yes** to confirm the chaincode deletion.

## Manage Chaincode Versions

(Hyperledger Fabric v2.x) Each chaincode that you deploy or upgrade consists of a chaincode package and a chaincode definition.

1. Go to the console and select the Channels tab.
2. Select the channel that you want to inspect and then click **Deployed Chaincodes**.

   The deployed chaincodes summary table lists the names and versions of the chaincodes deployed to the channel.

3. Click the **More Actions** menu icon for a chaincode and then click **View Chaincode Definition** to see the chaincode definition, including the endorsement policy and private data collections.
4. Click a package ID. The Installed Peers Summary page is displayed, showing which peers the package is installed on. You can click the peer to view more information about it.
5. Click the Deployed on Channels pane to see all of the channels the chaincode is deployed on. You can click a channel to view more information about it.

   From this pane, you can click Deploy on a New Channel to deploy the chaincode package to a different channel, or to the same channel using a different chaincode definition.

## Upgrade a Chaincode

(Hyperledger Fabric v2.x) If a developer modifies a chaincode's source, then you'll need to deploy it to a new version of the chaincode.

You can deploy different versions of the same chaincode on different channels.

You must be an administrator to perform this task. If you use the console, the upgrade process includes both approving and committing the upgraded chaincode. You can also use the REST API to upgrade a deployed chaincode by using the same calls that you use to install, approve, and commit a chaincode. For more information, see REST API for Oracle Blockchain Platform on Oracle Cloud Infrastructure (Gen 2).

1. Go to the console and select the Channels tab.

The Channels tab is displayed and the table lists all of the channels on the network.

2. Click the channel where the chaincode that you want to upgrade is deployed, and then click **Deployed Chaincodes**.
3. Locate the chaincode that you want to upgrade, click **More Actions**, and select **Upgrade**.

   The Upgrade Chaincode page is displayed.
4. Specify a **Chaincode Version** and select a **Package ID** to use in the chaincode definition.
5. If the chaincode requires initialization, select **Init-required**.

   If **Init-required** is selected, the client application must invoke the `Init` function explicitly, by specifying the `isInit` flag, before calling any other function.
6. If required, enter an endorsement policy and private data collections, and then click **Upgrade**.

   The chaincode is upgraded and deployed.

## What Are Private Data Collections?

(Hyperledger Fabric v2.x) Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data and to prevent the other organizations on the channel from seeing the data. Private data is distributed peer to peer and not by blocks, so the transaction data is kept confidential from the ordering service. Collections help you reduce the number of channels and their required maintenance on your network.

The primary components in a private data collection are:

- The private data that you specify in your private data collection definition. Private data is sent with the gossip protocol from peer to peer within the organizations that you specify in your policy. Private data is stored in a private database on the peer. The ordering service isn't used and can't see the private data.
- A hash of the data, which is endorsed, ordered, and written to each peer on the channel. This hash is evidence of the transaction and can be used for audit purposes.

When you deploy a chaincode, you can associate it with one or more private data collections.

## Add Private Data Collections

(Hyperledger Fabric v2.x) You can add private data collections to channels. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data within a transaction and to prevent the other organizations on the channel from seeing the data.

If you're going to use private data collections across the organizations in your network, then you need to configure anchor peers. Anchor peers facilitate private data gossip among the organizations. See Add an Anchor Peer.

You specify the private data collections when you deploy the chaincode.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to deploy and begin the deployment process.
3. Expand the Private Data Collections section and add the collection definition as needed.

| Field | Description |
|---|---|
| Collection Name | Enter the collection's name. You'll reference this name in the chaincode. |
| Policy | Create the policy to specify which organizations are included in the collection and which peers can store the private data.<br>Each member listed in the policy must be included in an `OR` signature policy list.<br>To support read/write transactions, the private data distribution policy must contain more organizations than the chaincode endorsement policy because peers must have the private data to endorse transactions. For example, in a channel with ten organizations, five of the organizations are included in a private data collection policy, but the endorsement policy requires three organizations to endorse a transaction. |
| Peers Required | Enter the number of peers that each endorsing peer must distribute private data to before the peer signs the endorsement and returns the proposal response.<br>Set this value to 1 or more peers to ensure the following:<br>• Redundancy of the private data on multiple peers in the network.<br>• Availability of the private data if the endorsing peers become unavailable.<br>Note that setting this value to 0 means that distribution isn't required. However, if the **Max Peer Count** field is set to greater than 0, private data distribution might still occur. |
| Max Peer Count | Enter the maximum number of peers that the current endorsing peer attempts to distribute the data to. This is to ensure redundancy so that peers are available between endorsement time and commit time to pull the private data if an endorsing peer isn't available.<br>If you set this value to 0, the private data isn't distributed at the time of endorsement. This causes private data pulls against the endorsing peers on all authorized peers at commit time. |

| Field | Description |
|---|---|
| Block to Live | Enter the length in number of blocks that you want data to reside on the private database. The data is purged when the number of blocks is reached.<br>Set this value to 0 if you never want to purge the data.<br>Note that a peer can fail to pull private data from another peer if a private data collection's `blocktolive` value is less than 10, and its `requiredPeerCount` and `maxPeerCount` values are less than the total number of peers in the channel. This is a known Hyperledger Fabric issue. |
| Endorsement Policy | Optionally, specify an endorsement policy for the collection that overrides the chaincode's endorsement policy.<br>Choose a **Policy Type** of either **Signature Policy** or **Channel Config policy** to use a signature policy or an existing channel configuration policy.<br>For **Policy**, specify an expression that represents the endorsement policy. For more information, see Endorsement policies in the Hyperledger Fabric documentation. |
| Member Only Read | Select to automatically prevent members of organizations that are not part of the collection from reading private data. |
| Member Only Write | Select to automatically prevent members of organizations that are not part of the collection from writing private data. |

4. Click **Add New Collection** and your collection's information is displayed in the private data collection table.
5. If needed, specify other collections.
6. Complete the other fields on the Deploy Chaincode page as needed.
7. Click **Deploy**.

# View Private Data Collections

(Hyperledger Fabric v2.x) You can view information about a chaincode's private data collections.

After you deploy a chaincode, you might need to view its private data collections to see how they were defined.

You can’t modify the private data collections for a deployed chaincode. To change the private data collections, upgrade the chaincode and specify new private data collections.

1. Go to the console and select the **Channels** tab.
2. Click the name of the channel where the chaincode is deployed, and then click **Deployed Chaincodes** to open the Deployed Chaincodes Summary page.
3. Click the **More Actions** menu icon for the desired deployed chaincode.

4. Click **View Chaincode Definition**.
5. On the Chaincode Definition window, expand **Private Data Collection**, and then locate the collection that you want to view.

# Deploy and Manage Chaincodes on Hyperledger Fabric v1.4.7

This topic contains information to help you deploy (install, instantiate, upgrade, and enable in the REST proxy), monitor, and find information about the chaincodes on the network.

**Topics**



## Typical Workflow to Deploy Chaincodes

(Hyperledger Fabric v1.4.7) Here are the common tasks for deploying chaincodes.

You must be an administrator to perform these tasks.

| Task | Description | More Information |
|---|---|---|
| Use the wizard to fully or partially deploy a chaincode | For testing, use Quick Deployment to perform the deployment in one step, using default settings.<br>For production, use Advanced Deployment to specify the deployment settings such as which peers to install the chaincode on and the endorsement policy you want to use. With Advanced Deployment you can instantiate the chaincode and enable it in the REST proxy now or later. | Use Quick Deployment<br>Use Advanced Deployment |

| Task | Description | More Information |
|---|---|---|
| Instantiate a chaincode | Instantiate the chaincode after you've installed it. | Instantiate a Chaincode |
| Upgrade the chaincode | Upload and instantiate a newer version of a chaincode, or pick an older version of the chaincode to use. | Upgrade a Chaincode |

# Use Quick Deployment

(Hyperledger Fabric v1.4.7) Use the quick deployment option to perform a one-step chaincode deployment. This option is recommended for chaincode testing.

The quick deployment uses default settings, installs the chaincode on all peers in the channel, instantiates the chaincode using the default endorsement policy, and enables the chaincode in the REST proxy.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7).
- You can use the advanced deployment option to put your chaincode into production on the network. See Use Advanced Deployment.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Quick Deployment**.

   The Deploy Chaincode (Quick) page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field enter a string value to specify the chaincode's version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).
5. Review the other default settings and modify them as needed.
6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy.
7. Click **Submit**.

The chaincode is installed on the channel's peers, instantiated, and enabled in the REST proxy. The deployed chaincode's name is displayed in the Chaincode tab's table.

## Use Advanced Deployment

(Hyperledger Fabric v1.4.7) Use the advanced deployment option to specify the parameters required to deploy a chaincode into a production environment. For example, you'll specify which peers to install the chaincode on and the endorsement policy to use.

With the advanced deployment wizard, you'll install the chaincode on the peers you select.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7).
- You can use the quick deployment option for chaincode testing. Quick deployment is a one-step deployment that uses default settings, installs the chaincode on all peers in the channel, and instantiates the chaincode using a default endorsement policy. See Use Quick Deployment.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Advanced Deployment**.

   The Deploy Chaincode (Advanced) Step 1 of 3: Install page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field, enter the chaincode's version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).
5. Select one or more network peers to install the chaincode onto. To provide high availability, Oracle suggests that you choose the appropriate number of peers from each partition. Also, the peers you choose must be joined to the channel that you'll instantiate the chaincode on.

6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy. Click **Next**.

   The chaincode is installed and the Deploy Chaincode (Advanced) Step 2 of 3: Instantiate page is displayed.

7. Decide if you want to instantiate the chaincode now or later.

   - Click **Close** to close the wizard and instantiate later.
   - To instantiate now, select the channel to instantiate the chaincode on and the peers to instantiate the chaincode to. If required, enter initial parameters, an endorsement policy, transient map, and private data collections. Note the following information:
     - Instantiation compiles, builds, and initializes the chaincode on the peers.
     - If you leave the endorsement policy blank, then Oracle Blockchain Platform uses the default endorsement policy. The default endorsement policy gets an endorsement from any peer on the network.
     - When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

     Click **Next**.

   The chaincode is instantiated.

## Instantiate a Chaincode

(Hyperledger Fabric v1.4.7) Instantiating a chaincode compiles, builds, and initializes the chaincode on the peers where the chaincode is installed. When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

Note the following information:

- You must install the chaincode on the required peers before you can instantiate it.
- If you're working on a channel that contains multiple members and have instantiated the chaincode on one member, then you don't have to instantiate the chaincode on the other members where you installed the same chaincode. In such cases, the chaincode is already instantiated and running on all members on the channel.
- You can instantiate more than one chaincode on a channel.
- The process to instantiate the sample chaincodes is different than the instantiation process described in this topic. See Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7).
- After you instantiate the chaincode, then you can optionally enable it in the REST proxy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click the arrow to expand the chaincode's version list.
3. Locate the chaincode version and click its **More Actions** menu, and select **Instantiate**.

   The Instantiate Chaincode dialog is displayed.

4. Enter information about where and how to instantiate the chaincode.

| Field | Description |
|---|---|
| Channel | Select the channel for the chaincode to run on. |

| Field | Description |
|---|---|
| Peers | Select the peer or peers you want to use the chaincode. This list shows the peers that you installed the chaincode onto. |
| Initial Parameter | Enter the input parameters that you want to pass to the chaincode. Go to the chaincode to find the initial parameters values. |
| Endorsement Policy | In this section, specify the number and role of members required to endorse the chaincode.<br>If you don't specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network. |
| Transient Map | The data that is passed into the chaincode is the transaction payload and the transient map. The transaction payload is recorded in the ledger and is visible to anyone who can access the ledger through the query system chaincode. Use a transient map to pass private data such as keys that you don't want stored in the ledger.<br>In this section, provide the required keys and values. The information you provide is maintained on the peer node and is sent to the chaincode when a transaction is executed.<br>If you're adding private data collections, then specify a transient map to pass the private data from the client to the peers for endorsement. |
| Private Data Collections | In this section, add one or more private data collections. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel you instantiate the chaincode on. |

5. Click **Instantiate**.

   The chaincode is instantiated.

6. To confirm that the chaincode was instantiated, go to the Channels tab and click the name of the channel that you instantiated the chaincode on. Go to the Instantiated Chaincodes tab and confirm that the chaincode is listed in the summary table.

## Specify an Endorsement Policy

(Hyperledger Fabric v1.4.7) You can add an endorsement policy when you instantiate a chaincode. An endorsement policy specifies the members with peers that must approve, or properly endorse, a chaincode transaction before it's added to a block and submitted to the ledger.

Endorsement guarantees the legitimacy of a transaction. When you instantiate a chaincode on a channel, you can specify an endorsement policy. If you don't specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

A member's endorsing peers must have ReaderWriter permissions on the channel. When a transaction is processed, each endorsing peer returns a signed read-write set. After the client has enough endorsements to meet the endorsement policy requirements, then the client bundles the common read-write set with the signature from the endorsing peers and sends everything to the ordering service, which orders and commits the transactions into blocks and then to the ledger.

You can go to the Channels tab to view an instantiated chaincode's endorsement policy. See View an Endorsement Policy. You can't modify an instantiated chaincode's endorsement policy. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Endorsement Policy section. Click **Add Identity** to add members to the policy as needed.

| Field | Description |
|---|---|
| MSP ID | From the dropdown menu, select the endorser peer's organization. |
| Role | Select the corresponding peer role required by the endorsement policy. Usually this will be member. You can find a peer's role by viewing its configuration information. If Node OU is enabled, there are three options: admin, member, and peer. The peer option is for use with Node OU. |
| Policy Expression Mode | In most cases, you'll use **Basic**. Select **Advanced** to provide an expression string. See the Hyperledger Fabric documentation for information about how to write a valid expression string. |
| Signed By | Select how many members with endorsing peers (peers with ReaderWriter permissions) on the channel must endorse the chaincode transactions to make them valid. |

4. Complete the other fields on the Instantiate Chaincode page as needed.
5. Click **Instantiate**.

## View an Endorsement Policy

(Hyperledger Fabric v1.4.7) You can view an instantiated chaincode's endorsement policy.

You might need to view an instantiated chaincode's endorsement policy to see how it was set up, how you need to choose transaction endorsers based on the policy, or to help resolve an endorsement failure.

You can't modify the endorsement policy for an instantiated chaincode. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.
2. Locate the chaincode that you want to view endorsement policy information for and expand it in the table.
3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.
4. In the Instantiated on Channels tab, locate the channel that you want, click **More Actions**, and select **View Endorsement Policy**.

   The Chaincode Endorsement Policy page is displayed.

## Find Information About Chaincodes

(Hyperledger Fabric v1.4.7) You can find information about the chaincodes in your network. For example, how many peers the chaincode is installed on and if the chaincode has been instantiated.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes and versions installed on the network.

2. In the chaincode table, locate the chaincode that you want information for and expand it to see information about its versions, path, how many peers it's installed on, and how many channels it's instantiated on.

   Note the following information:

   - When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Chaincodes tab.
   - If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. To list the chaincode, start at least one peer node that has the chaincode installed on it.

3. Use the chaincode table as a starting point to perform chaincode-related tasks, such as instantiate, enable it in the REST proxy, upgrade to a new version, and delete the chaincode.

## Manage Chaincode Versions

(Hyperledger Fabric v1.4.7) Each chaincode that you install or upgrade has a version number.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes installed on the network.

2. Locate the chaincode that you want version information for and expand it to see a list of versions.
3. Click a version number. The Chaincode Version Information page is displayed.
4. Click the Installed on Peers pane to see which peers the chaincode is installed on. You can click the peer to view more information about it.
5. Click the Instantiated on Channels pane to see which channels the chaincode is instantiated on. You can click a channel to view more information about it.

   From this pane, you can also instantiate a specific version of the chaincode version. If the chaincode was instantiated on a channel, then you can view its endorsement policy.

   Note that you can instantiate different versions of a chaincode on different channels.

6. Click the Private Data Collections pane to view the private data collections that were added when the chaincode was instantiated.

## Upgrade a Chaincode

(Hyperledger Fabric v1.4.7) If a developer modifies a chaincode's source, then you'll need to deploy it to a new version of the chaincode. If needed, you can revert back to an older version of a chaincode.

You can instantiate different versions of the same chaincode on different channels.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists all of the chaincodes installed on the network.

2. Locate the chaincode that you want to upgrade, click **More Actions**, and select **Upgrade**. The **More Actions** button only displays for chaincodes that have been instantiated.

   The Upgrade Chaincode Step 1 of 2: Select a version page is displayed.

3. Select a version source. Note the following information:

   - Click **Select from existing versions** if you want to upgrade to a version that is already on the network. You might choose this option because the most current chaincode version contains errors and you need to temporarily use an older version until the chaincode can be fixed. Because the older version is on your system, the chaincode is already installed on the peers.
   - Choose **Install a new version** to upload the chaincode file. In the **Version** field enter a version number and in the **Target Peers** field, select the peers to install the chaincode on. In the **Chaincode Source** field, click **Upload Chaincode File** and browse for the chaincode ZIP file to upload.

4. Click **Next**.

   The Upgrade Chaincode Step 2 of 2: Upgrade page is displayed.

5. Decide if you want to instantiate the chaincode version now or later.

   - Click **Close** to close the wizard and upgrade later.
   - To upgrade now, select the channel to upgrade the chaincode on and the peers to instantiate the chaincode to. If required, enter initialize parameters, an endorsement policy, and transient map. See Specify an Endorsement Policy. Click **Next**.

   The chaincode is upgraded.

# What Are Private Data Collections?

(Hyperledger Fabric v1.4.7) Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data and to prevent the other organizations on the channel from seeing the data. Private data is distributed peer to peer and not by blocks, so the transaction data is kept confidential from the ordering service. Collections help you reduce the number of channels and their required maintenance on your network.

The primary components in a private data collection are:

- The private data that you specify in your private data collection definition. Private data is sent with the gossip protocol from peer to peer within the organizations that you specify in your policy. Private data is stored in a private database on the peer. The ordering service isn't used and can't see the private data.
- A hash of the data, which is endorsed, ordered, and written to each peer on the channel. This hash is evidence of the transaction and can be used for audit purposes.

When you instantiate a chaincode, you can associate it with one or more private data collections. Also when you instantiate a chaincode, you should specify a transient map to pass the private data from the client to the peers for endorsement. The collection definition specifies who can persist data, how many peers the data is distributed to, how many peers

are required to disseminate the private data, and how long the private data is persisted in the private database.

## Add Private Data Collections

(Hyperledger Fabric v1.4.7) You can add private data collections to channels. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data within a transaction and to prevent the other organizations on the channel from seeing the data.

If you're going to use private data collections across the organizations in your network, then you need to configure anchor peers. Anchor peers facilitate private data gossip among the organizations. See Add an Anchor Peer.

You specify the private data collections when you instantiate the chaincode.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Private Data Collections section and add the collection definition as needed.

| Field | Description |
|---|---|
| Collection Name | Enter the collection's name. You'll reference this name in the chaincode. |
| Policy | Create the policy to specify which organizations are included in the collection and which peers can store the private data.<br>Each member listed in the policy must be included in an `OR` signature policy list.<br>To support read/write transactions, the private data distribution policy must contain more organizations than the chaincode endorsement policy because peers must have the private data to endorse transactions. For example, in a channel with ten organizations, five of the organizations are included in a private data collection policy, but the endorsement policy requires three organizations to endorse a transaction. |

| Field | Description |
|---|---|
| Peers Required | Enter the number of peers that each endorsing peer must distribute private data to before the peer signs the endorsement and returns the proposal response.<br>Oracle recommends that you set this value to 1 or more peers to:<br>• Ensure redundancy of the private data on multiple peers in the network.<br>• Ensure that private data is available if the endorsing peers become unavailable.<br>Note that setting this value to 0 means that distribution isn't required. However, if the **Max Peer Count** field is set to greater than 0, then private data distribution might still occur. |
| Max Peer Count | Enter the maximum number of peers that the current endorsing peer attempts to distribute the data to. This is to ensure redundancy so that peers are available between endorsement time and commit time to pull the private data if an endorsing peer isn't available.<br>If you set this value to 0, then the private data isn't distributes at the time of endorsement. This causes private data pulls against the endorsing peers on all authorized peers at commit time. |
| Block to Live | Enter the length in number of blocks that you want data to reside on the private database. The data is purged when the number of blocks is reached.<br>Set this value to 0 if you never want to purge the data.<br>Note that a peer can fail to pull private data from another peer if a private data collection's `blocktolive` value is less than 10, and its `requiredPeerCount` and `maxPeerCount` values are less than the total number of peers in the channel. This is a known Hyperledger Fabric issue. |

4. Click **Add New Collection** and your collection's information is displayed in the private data collection table.
5. If needed, specify other collections.
6. Complete the other fields on the Instantiate Chaincode page as needed.
7. Click **Instantiate**.

# View Private Data Collections

(Hyperledger Fabric v1.4.7) You can view information about a chaincode's private data collections.

After you instantiate a chaincode, you might need to view its private data collections to see how they were defined.

You can’t modify the private data collections for an instantiated chaincode. To change the private data collections, upgrade the chaincode and specify new private data collections.

1. Go to the console and select the **Chaincodes** tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.

2. Locate the chaincode that you want to view private data collections for and expand it in the table.

3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.

4. In the Private Data Collections tab, locate the collection that you want to view.

9

# Develop Blockchain Applications

Blockchains require smart contracts (chaincode) to update the ledger. In addition, you will also require a client application that utilizes either the Oracle Blockchain Platform REST API or native Hyperledger Fabric SDK to interact with the blockchain directly. There are other operational and administrative tasks to consider, namely the creation of peers and channels and installation of chaincode.

**Topics**



## Before You Develop an Application

Before you write an application, download and use the sample applications, and ensure that you've the correct certificates and privileges to run an application.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See:

- What Are Chaincode Samples?
- Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v2.x)
- Explore Oracle Blockchain Platform Using Samples (Hyperledger Fabric v1.4.7)

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write applications. Read the *Key Concepts* and *Tutorials* sections before you write your own application: Hyperledger Fabric documentation.

**Prerequisites for Application Development**

A user ID and password for the application user must exist in Oracle Identity Cloud Service. Depending on the functions in the application, this user must have the following prerequisites:

- To install and deploy chaincode:
  - You must have administrative access in order to install or deploy chaincode.
  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer and orderer nodes crypto folders.
  - You must export the admin credentials similarly to how you exported the certificates (from the action menu, select **Export Admin Credential**). This will download a ZIP file containing the signed certificate and keystore files that need to be placed in your application in the peer and orderer nodes crypto folders.

- To run operations against an installed and deployed chaincode:
  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer node crypto folders.
  - You must export the tlscacerts certificate for the orderer node as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service so that it can be placed in your application.
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.
- To run functions against a REST API endpoint:
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.

# Use the Hyperledger Fabric SDKs to Develop Applications

Applications use a software development kit (SDK) to access the APIs that permit queries and updates to the ledger. You can install and use the Hyperledger Fabric SDKs to develop applications for Oracle Blockchain Platform.

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. See REST API for Oracle Blockchain Platform.

However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

**SDK Versions**

Multiple versions of the Hyperledger Fabric SDKs are available. Use the version of the SDK that matches the version of Hyperledger Fabric that your instance is based on.

**Installing the Hyperledger Fabric SDK for Node.js**

Information about how to use the Fabric SDK for Node.js can be found here: Hyperledger Fabric SDK for Node.js documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Node.js SDK by using the link on this tab.
- (Hyperledger Fabric v1.4.7) If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric v1.4.7 SDKs to Work with Oracle Blockchain Platform.

**Installing the Hyperledger Fabric SDK for Java**

Information about how to use the Fabric SDK for Java can be found here: Hyperledger Fabric SDK for Java documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Java SDK by using the link on this tab.
- (Hyperledger Fabric v2.x) If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric v2.x SDKs to Work with Oracle Blockchain Platform.
- (Hyperledger Fabric v1.4.7) If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric v1.4.7 SDKs to Work with Oracle Blockchain Platform.

Install a build tool such as Apache Maven.

**Structuring your Application**

Your Java application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act on the
orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act on the
peer node
    /src
      chaincode.go if installing and deploying chaincode to the blockchain
  /java
    pom.xml or other build configuration files
    /resources
      Any resources used by the Java code, including artifacts such as the
endorsement policy yaml file and blockchain configuration properties
    /src
      Java source files
```

Your Node.js application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act on the
orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act on the
peer node
    /src
      chaincode.go if installing and deploying chaincode to the blockchain
  /node
    package.json file
    application.js
    /app
      Any javascript files called by the application
      /tools
```

### Running the application

You're now ready to run and test the application. In addition to any status messages returned by your application, you can check the ledger in the Oracle Blockchain Platform console to see your changes:

1. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
2. In the channel's **Ledger** pane, view the chaincode's ledger summary.

## Update the Hyperledger Fabric v2.x SDKs to Work with Oracle Blockchain Platform

There's an incompatibility between an OCI infrastructure component and the Java SDK provided with Hyperledger Fabric v2.x. Follow the steps in this topic to correct this problem.

### Methods of updating the Hyperledger Fabric SDK

There are two ways of updating the SDK:

- Download the modified package from the console. We've created an updated `grpc-netty-shaded-1.38.0.jar` file, which is the module referenced by the Java SDK that requires modifications.
- Build the package manually, as described in this topic.

To download the `grpc-netty-shaded-1.38.0.jar` file, click the console's **Developer Tools** tab, and then select the **Application Development** pane.

### Manually building the package

For the `fabric-sdk-java` project, complete the following steps to rebuild the `grpc-netty-shaded` package to connect the peers and orderers with the `grpcs` client (via TLS). The `grpc-netty-shaded` package is a sub-project of `grpc-java`.

1. Install project dependencies:

   ```
   mvn install
   ```

2. Download the `grpc-java` source code:

   ```
   git clone https://github.com/grpc/grpc-java.git
   ```

3. Find the `grpc` version that your `fabric-sdk-java` uses, and check out the code. In the `grpc-java` directory, check out the version of `grpc` that `fabric-sdk-java` uses:

   ```
   cd grpc-java && git checkout v1.38.0
   ```

4. Change the code to avoid an `alpn` error from the server side. Create a `grpc-java-patch` patch file with the following contents:

   ```
   diff --git a/netty/src/main/java/io/grpc/netty/
   ProtocolNegotiators.java b/netty/src/main/java/io/grpc/netty/
   ProtocolNegotiators.java
   ```

```
index 19d3e01b7..ebc4786a3 100644
— a/netty/src/main/java/io/grpc/netty/ProtocolNegotiators.java
+++ b/netty/src/main/java/io/grpc/netty/ProtocolNegotiators.java
@@ -611,7 +611,8 @@ final class ProtocolNegotiators {
SslHandshakeCompletionEvent handshakeEvent =
(SslHandshakeCompletionEvent) evt;
if (handshakeEvent.isSuccess()) {
SslHandler handler = ctx.pipeline().get(SslHandler.class);

    if (sslContext.applicationProtocolNegotiator().protocols()
    + if (handler.applicationProtocol() == null
    + || sslContext.applicationProtocolNegotiator().protocols()
    .contains(handler.applicationProtocol())) {
    // Successfully negotiated the protocol.
    logSslEngineDetails(Level.FINER, ctx, "TLS negotiation succeeded.",
null);
```

5. Apply the patch:

   ```
   git apply grpc-java.patch
   ```

6. Build the project to generate the target patched package. Use gradle to build the `grpc-java-shaded` project:

   ```
   cd netty/shaded
   gradle build -PskipAndroid=true -PskipCodegen=true
   ```

   After the build completes, the target patched `.jar` package is available at `grpc-java/netty/build/libs/grpc-netty-shaded-1.38.0.jar`.

7. Add the patched package into your Maven local repository.

   Replace the original `grpc-netty-shaded .jar` package with the patched package in either of the following two ways:

   - Use Maven to install the package by local file:

     ```
     mvn install:install-file -Dfile=grpc-netty-shaded-build/grpc-netty-
     shaded-1.38.0.jar -DgroupId=io.grpc -DartifactId=grpc-netty-shaded -
     Dversion=1.38.0 -Dpackaging=jar
     ```

     You must keep the target `groupid`, `artifactid`, and `version` the same as the package you want to replace.

   - Manually replace your package. Go to the local Maven repository, find the directory where the target package is located, and replace the package with patched package.

8. Run the project.

# Update the Hyperledger Fabric v1.4.7 SDKs to Work with Oracle Blockchain Platform

There's an incompatibility between an OCI infrastructure component and the Node.js and Java SDKs provided with Hyperledger Fabric v1.4.7. Follow the steps in this topic to correct this problem.

**Methods of updating the Hyperledger Fabric SDKs**

There are two ways of updating the SDK:

- Using Oracle scripts to download and install the Node.js SDK or Java SDK which will patch the code as it installs.
- Manually as described in this topic.

To use the scripts, on the console's **Developer Tools** tab, select the **Application Development** pane. The links to download both the Node.js SDK and Java SDK have updates built in which will patch the code as it installs.

- Fabric Java SDK: We've created an updated `grpc-netty-1.23.0.jar` file, which is the module referenced by the Java SDK which requires modifications.
- Fabric Node.js SDK: We have created the `npm_bcs_client.sh` script to replace the standard Fabric `npm install` operations that users would perform to download and install the Node.js Fabric client package. The script runs the same npm command, but it also patched the needed component and rebuilds it.

**Manually updating the Fabric Node.js SDK**

Do the following to rebuild the `grpc-node` module to connect the peers and orderers with grpcs client (via TLS).

1. Install `fabric-client` without executing the grpc module's build script:

   ```
   npm install --ignore-scripts fabric-client
   ```

2. On Windows, you need to disable ALPN explicitly
   - Update `node_modules/grpc/binding.gyp` by changing:

     ```
     '_WIN32_WINNT=0x0600'
     ```

     to

     ```
     '_WIN32_WINNT=0x0600','TSI_OPENSSL_ALPN_SUPPORT=0'
     ```

   - Due to the issue outlined in https://github.com/nodejs/node/issues/4932, to build `grpc-node` on Windows, you must first remove `<node_root_dir>/include/node/openssl/`. Run the following to find your `<node_root_dir>`:

     ```
     node-gyp configure
     ```

3. Rebuild `grpc`

```
npm rebuild --unsafe-perm --build-from-source
```

You can now install any other modules you need and run the project.

**Manually updating the Fabric Java SDK**

For `fabric-sdk-java`, do the following steps to rebuild the `grpc-netty` package to connect the peers and orderers with grpcs client (via tls). `grpc-netty` is a sub-project of `grpc-java`.

1. Install project dependencies:

```
mvn install
```

2. Download grpc-java source code:

```
git clone https://github.com/grpc/grpc-java.git
```

3. Find the `grpc` version that your `fabric-sdk-java` uses, and checkout the code. Different versions of `fabric-sdk-java` may use different version of `grpc`. Check `pom.xml` to find out what `grpc` version your `fabric-sdk-java` uses. For example, `fabric-sdk-java 1.4.11` uses `grpc-java 1.23.0` as found in its `pom.xml`: https://github.com/hyperledger/fabric-sdk-java/blob/v1.4.11/pom.xml.

   In the `grpc-java` directory, checkout the version of `grpc` that `fabric-sdk-java` uses:

```
git checkout -b v1.23.0
```

4. Change the code to avoid an `alpn` error from the server side.

   - Change the target code of `grpc-java_root/netty/src/main/java/io/grpc/netty/ProtocolNegotiators.java`
   - In the function `userEventTriggered0` change:

```
if (NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
```

   to

```
if (handler.applicationProtocol() == null ||
NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
```

   The code will look similar to:

```
    @Override
    protected void userEventTriggered0(ChannelHandlerContext ctx,
Object evt) throws Exception {
      ...
          if (handler.applicationProtocol() == null ||
NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
            // Successfully negotiated the protocol.
            logSslEngineDetails(Level.FINER, ctx, "TLS negotiation
succeeded.", null);
```

```
        ...
    }
```

5. Build the project to generate the target patched package. Use gradle to build the `grpc-java` project. Or you can just rebuild the `grpc-netty` sub-project in the grpc netty directory `gradle build`.

   After the build is done, you can get the target patched jar package in the directory `grpc-java\netty\build\libs\grpc-netty-1.23.0.jar`.

6. Add the patched package into your Maven local repository.

   Replace official `grpc-netty` jar package with the patched package in either of the following two ways:

   - Use Maven to install the package by local file:

     ```
     mvn install:install-file -
     Dfile=local_patched_grpc_netty_package_root/grpc-
     netty-1.23.0.jar -DgroupId=io.grpc -DartifactId=grpc-netty -
     Dversion=1.23.0 -Dpackaging=jar
     ```

     You must keep the target `groupid`, `artifactid`, and `version` the same as the package you want to replace.

   - Manually replace your package. Go to the local Maven repository, find the directory where the target package is located, and replace the package with patched package.

7. Run the project.

# Use the REST APIs to Develop Applications

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

Any application using the REST APIs requires the following:

- The chaincode name and version.
- The REST server URL and port, and the user ID and password for the REST node.
- Functions to invoke transactions against or query the ledger.

See REST API for Oracle Blockchain Platform for information on the existing operations, including examples and usage syntax.

### Structuring your Application

Your REST API application should be structured similar to the following:

```
/Application
  /artifacts
    /crypto
      /orderer
```

```
        Contains the certificates required for the application to act on the
orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act on the
peer node
    /src
  /REST
    Application script containing REST API calls
```

# Make Atomic Updates Across Chaincodes and Channels

You can use atomic transactions to complete multiple transactions across channels and chaincodes in an atomic manner.

An atomic transaction is an indivisible series of data operations that either all succeed, or none succeed.

Atomic transactions can be useful in complex situations where multiple chaincodes are deployed to separate channels. You can use atomic transactions to maintain data consistency while running multiple blockchain transactions, even if a network or system failure occurs. Oracle Blockchain Platform supports atomic transactions by using the two-phase commit protocol, where an initial phase where each data operation is prepared is followed by a phase where each data operation is actually committed.

Atomic transactions work at the application level. Typically you do not need to change existing chaincode logic to support atomic transactions. Because one or more additional arguments are added by the atomic transactions framework, make sure that any existing chaincode does not perform strict checks on the number of arguments passed in the chaincode method. Atomic transactions are supported by the following REST API endpoint:

- `restproxy/api/v2/atomicTransactions`

The REST API endpoint prepares the transactions as defined by your chaincode, and then uses built-in chaincode functions to either to commit all of the transactions, or to roll back all of the transactions if there are any errors during the prepare phase. For more information about the REST endpoints to use to implement atomic transactions, see Atomic Transactions REST Endpoints.

Each atomic transaction is composed of two or more blockchain transactions. The result (the `returnCode` value) of the atomic transaction is either `Success` or `Failure`. In an atomic transaction, each requested blockchain transaction is split into two distinct operations: a prepare phase and then either a commit or a rollback phase.

- In the prepare phase, each transaction is endorsed as usual, but instead of being finalized, the changes are staged and the values are locked to prevent other transactions from modifying the staged values.
- If the prepare phase is successful for each blockchain transaction, then the transactions are endorsed and committed by using built-in chaincode. The previously locked values are unlocked, and the result of the atomic transaction is `Success`.
- If the prepare phase fails for any blockchain transaction, then all other transactions where the prepare phase succeeded are rolled back, again by using built-in chaincode. The staged changes are removed and the previously locked values are unlocked. The result of the atomic transaction is `Failure`.

Because the atomic transactions works by locking keys, you might receive a `Two_Phase_Commit_Lock` error if a different transactions attempts to modify a key that is locked by a currently active atomic transaction that is prepared. This can occur in one of the following two scenarios:

- An atomic transaction is still in the prepared phase, and a different transaction attempts to modify a key that was locked by the prepared transaction. In this case, the system is working as designed. If you encounter this error, retry the second transaction. This is analogous to how applications handle phantom read errors or multi-version concurrency control (MVCC) errors.
- The `GlobalStatus` value returned by the atomic transaction is `HeuristicOutcome`. In this case, an atomic transaction operation was canceled because one of the commit operations failed. This is a rare occurrence and might need to be resolved manually. One side effect of a heuristic outcome is that some keys may be left locked by transactions which failed to be committed or rolled back. In this case, use the following REST API endpoint to unlock the atomic transaction:
  - `restproxy/api/v2/atomicTransactions/{globalTransactionId}`

  For more information about the REST endpoint to use to unlock atomic transactions, see Unlock Atomic Transaction.

**Scenario: Explore Atomic Transactions Using Samples**

Consider the following example, which uses two of the sample chaincodes that are included with Oracle Blockchain Platform, Balance Transfer and Marbles. The Balance Transfer sample represents two parties with the ability to transfer funds between account balances. The Marbles sample lets you create marbles and exchange them between owners. You could use individual (non-atomic) transactions to buy a marble by exchanging funds in the Balance Transfer chaincode and changing the ownership of the marble in the Marbles chaincode. However, if an error occurs with one of those transactions, the ledger might be left in an inconsistent state: either the funds were transferred but not the marble or the marble is transferred but not paid for.

In this scenario, you can use the existing chaincode with the REST API endpoints that support atomic transactions. The exchange of funds and the transfer of ownership of the marble must both succeed or both fail. If either transaction encounters an error, then neither transaction is committed. To explore this scenario, complete the following steps:

1. Install the Balance Transfer and Marbles samples on different channels. For more information on installing the samples, see Explore Oracle Blockchain Platform Using Samples.
2. In the Marbles sample, invoke the `Create a new marble` action to create a number of marbles for various marble owners.
3. Use the `Invoke Atomic Transaction` REST endpoint to complete atomic transactions that invoke both the Marbles and the Balance Transfer samples.

For example, the following transaction transfers a marble named `marble1` to Tom, and sends 50 coins from account `a` to account `b`.

```
{
 "transactions": [
   {"chaincode":"obcs-marbles","args":["transferMarble", "marble1",
"tom"],"timeout":0, "channel":"goods"},
   {"chaincode":"obcs-example02","args":["invoke", "a", "b",
```

```
"50"],"timeout":0, "channel":"wallet"}
 ],
 "isolationLevel": "serializable",
 "prepareTimeout": 10000,
 "sync": true
}
```

In the previous transaction, if both transactions succeed in the prepare phase, then both transactions are committed to the ledger. If there is an error with either transaction, then neither transaction is committed during the second phase. Instead, both transactions are rolled back. For example, if there are less than 50 coins in account `a`, then no money is taken from the account and no marble is transferred to Tom.

There is a known issue with the Marbles sample and the default value of the **Marble Owner** field. For more information, see: Known Issues for Oracle Blockchain Platform.

# Ethereum Interoperability

You can include Ethereum-based transactions in an atomic transaction workflow.

Growing use of public blockchains and tokenization capabilities across both public and permissioned blockchains drives the need for their interoperability. Common scenarios include asset exchange across different ledgers, business transactions on permissioned blockchains that are linked to cryptocurrency payments on public chains, publishing proof of a permissioned blockchain transaction on a public blockchain, and so on. To enable interoperability for these and other scenarios, Oracle Blockchain Platform provides interoperability with Ethereum and with any EVM-based networks that support standard web3 protocols. The interoperability function works by incorporating the Geth Ethereum client in the REST proxy and enabling it to orchestrate an optimized two-phase commit protocol that includes both Oracle Blockchain Platform and Ethereum/EVM transactions through a single REST API called `atomicTransactions`. You can use the `atomicTransactions` API to send multiple chaincode transactions for multiple Oracle Blockchain Platform channels, and can optionally add an Ethereum transaction that will run atomically with the Oracle Blockchain Platform transactions.

Unlike Oracle Blockchain Platform transactions, Ethereum transactions cannot be broken down into the prepare and commit phases of the two-phase commit protocol. To include Ethereum transactions as part of an atomic workflow, Oracle Blockchain Platform uses a last resource commit (LRC) optimization. After all of the Oracle Blockchain Platform transactions are in the prepared state, the Ethereum transaction is started. If the Ethereum transaction succeeds, then the Oracle Blockchain Platform transactions are committed. If the Ethereum transaction fails, then the Oracle Blockchain Platform transactions are rolled back.

Ethereum transactions have a concept of **finality**. An Ethereum transaction can run successfully but it does not achieve finality until it's part of a block that can't change. You can use the `finalityParams` parameters to control whether to check for finality and how long to wait for it, either in blocks or in seconds. Typically, if you wait for six blocks to be generated on the public Ethereum blockchain network (**Mainnet**), you can assume that transaction finality was achieved. In private Ethereum networks, typically you do not need to wait as long for finality.

### Transferring an NFT to an Ethereum network

The `atomicTransactions` API also supports interactions with smart contracts that are deployed on Ethereum networks. You can use this functionality to transfer non-fungible

tokens (NFTs) that were minted in Hyperledger Fabric chaincode on Oracle Blockchain Platform to an Ethereum or Polygon network, by invoking two transactions atomically. NFT attributes such as the token ID, price, and token history can also be passed from Oracle Blockchain Platform to Ethereum atomically. After you transfer an NFT from Oracle Blockchain Platform to Ethereum, the NFT can be listed on a public NFT marketplace.

To transfer an NFT from Oracle Blockchain Platform to Ethereum, you use two basic steps in one atomic transaction:

1. Burn the NFT on Oracle Blockchain Platform. Call the `burnNFT` method, to burn (delete) the NFT from the Hyperledger Fabric chaincode on Oracle Blockchain Platform. Oracle Blockchain Platform supports NFTs in enhanced versions of two standards, ERC-721 and ERC-1155, with the Blockchain App Builder tool. For more information on the burnNFT method, see the relevant topic for your environment:
   - `burnNFT` (ERC-721, TypeScript)
   - `BurnNFT` (ERC-721, Go)
   - `burnNFT` (ERC-1155, TypeScript)
   - `BurnNFT` (ERC-1155, Go)
2. Mint the NFT on Ethereum. Call a smart contract on the Ethereum or Polygon network to mint the NFT on that network, using the parameters returned by the `burnNFT` method. Sample versions of smart contracts written in the Solidity language for NFTs are available in the following archive file: solidity-smartcontracts-fab253.zip. The smart contracts, one for each of the enhanced token standards ERC-721 and ERC-1155, include a `mintNFT` method, which creates NFTs with custom properties such as price and token history, which can be fetched from the output of the `burnNFT` method in the previous step.
   For unsigned requests, if the custom properties are in the `ParamKeys` parameter and corresponding dynamic parameters are passed in the `params` parameter, the atomic transactions API can fetch the parameters from the `burnNFT` method and send them to the Ethereum smart contract. The `mintNFT` method takes the following arguments:
   - `to` – The Ethereum address for the account where the NFT will be minted.
   - `id` – The token ID of the NFT.
   - `price` – The price of the NFT.
   - `tokenHistory` – The history of the NFT from the Oracle Blockchain Platform chaincode.

   The smart contract requires that the token ID of the NFT must be a numeric string (a string that can be converted to an integer). For example a token ID can be `2` but not `token2`.

   The token URI of the NFT in the chaincode deployed on Oracle Blockchain Platform must follow a certain format to make it compatible with Solidity smart contracts:
   - ERC-1155: A URI for all token types that relies on ID substitution, such as `https://token-cdn-domain/{id}.json`.

- ERC-721: A URI where all tokens share a prefix (a base URI) followed by a token URI, such as `http://api.myproject.example.com/token/<tokenURI>`.

You can use the Remix IDE to generate an application binary interface (ABI) for the smart contract. The ABI can then be used with the `atomicTransactions` API. If you change any method in the smart contract, you must recompile the contract and generate the ABI again.

For more information about the parameters to use for Ethereum transactions in an atomic workflow, including an example of transferring an NFT to an Ethereum network, see Atomic Transactions REST Endpoints.

# Include Oracle Blockchain Platform in Global Distributed Transactions

Your application might need to make updates across the Oracle Blockchain Platform ledger and other repositories such as databases or other blockchain ledgers in an atomic fashion, where either all updates succeed or none do.

To enable atomic updates across multiple databases, developers use global transactions that are coordinated by distributed transaction coordinators such as Oracle WebLogic Server, Oracle Tuxedo, Oracle Transaction Manager for Microservices, JBoss Enterprise Application Platform, IBM WebSphere, and other systems. All of these systems rely on the X/Open XA protocol to orchestrate a two-phase commit process by using standard APIs that are provided by XA Resource Managers (RMs) for each database or other resource. Oracle Blockchain Platform supports two-phase commits and provides its own XA RM library, which external transaction coordinators can use to invoke XA-compliant APIs. These global transactions can also include a single non-XA resource (for example, a non-Oracle blockchain ledger or non-XA compliant database) by using a last resource commit optimization.

The XA specification is part of the X/Open Distributed Transaction Processing architecture, which defines a standard architecture that enables multiple application programs to share resources provided by multiple resource managers. The Java XA interface itself is defined as part of the Java platform. For more information on the Java XA interface, see Interface XAResource in the Java documentation.

Oracle Blockchain Platform provides a library that conforms to the XA specification and implements the standard Java interface for an XA resource manager. The library enables a client-side transaction manager to coordinate global transactions. A global transaction is a single unit of work that might include operations such as database updates and blockchain transactions, all of which must be committed atomically. In other words, all of the operations must succeed to be committed. If any operation that is part of the global transaction fails, then all operations are rolled back. The XA interface relies on the two-phase commit protocol, similar to the protocol supported by the atomic transactions REST endpoints. For more information about atomic transactions in Oracle Blockchain Platform, see Make Atomic Updates Across Chaincodes and Channels.

The XA implementation for Oracle Blockchain Platform is supplied as a Java library, downloadable from the **Developer Tools** tab on the **Application Development** pane of the Oracle Blockchain Platform console.

Full details on the library are included in the Javadoc information supplied in the downloadable file. The three key objects supported by the library are `OBPXAResource`, `OBPXADataSource`, and `OBPXAConnection`.

| Object | Purpose |
|---|---|
| `OBPXAResource` | This class implements the required APIs for a transaction manager to coordinate with Oracle Blockchain Platform as a resource manager for XA transactions. |
| `OBPXADataSource` | Use this object to get an instance of the `OBPXAConnection` and to specify the authentication and authorization credentials. |
| `OBPXAConnection` | Use this object to get an instance of the `OBPXAResource` object and to define the blockchain transactions to run as part of an XA transaction. |

To use the XA library with Oracle Blockchain Platform, the application must provide credentials for authentication and authorization of the requested operations. The library supports both basic authentication (user/password) and OAuth 2.0 access tokens, which you can configure when you create the `OBPXADataSource` instance. The two authentication methods are consistent with the authentication methods that you use with the Oracle Blockchain Platform REST proxy. For more information, see Authentication in the REST API documentation.

After you create an `OBPXADataSource` instance, you can use the `obpxaDataSource.getXAConnection()` method to get the `xaConnection` instance. To update authentication when using OAuth 2.0 access tokens, you can use the `getXAConnection` method, as shown in the following code:

```
OBPXAConnection xaConnection =
obpxaDataSource.getXAConnection(accessToken);      // get an XA
connection using an OAuth 2.0 access token
```

You can also use the `getXAConnection` method to update basic authentication.

```
OBPXAConnection xaConnection = obpxaDataSource.getXAConnection(user,
password);   // get an XA connection using username and password for
basic authentication
```

To define a blockchain transaction to be run as part of a global XA transaction, you use the following method:

```
public void createXAInvokeTransaction(Xid xid, OBPXAInvokeTxRequest
invokeTxRequest)
```

In this method, `xid` is a global transaction identifier and `invokeTxRequest` is the blockchain transaction to be run as part of the global XA transaction. To create an XA invoke transaction request, you use the following constructor method:

```
OBPXAInvokeTxRequest invokeTxRequest = new
OBPXAInvokeTxRequest(channel, chaincode, args);
```

In this constructor method, `channel` is the channel where the blockchain transaction will run, `chaincode` is the chaincode to use, and `args` includes the chaincode function and any arguments to use for the transaction.

The following snippet of code demonstrates creating an `OBPXADataSource` object, getting the `OBPXAConnection` instance, and then creating the transaction request and calling the `createXAInvokeTransaction` method.

```
OBPXADataSource obpxaDataSource = OBPXADataSource.builder()
                        .withHost(host)
                        .withPort(port)
                        .withBasicAuth(username, password)
                        .withRole(role)
                        .build();
.
.
.
OBPXAConnection obpxaConnection = obpxaDataSource.getXAConnection();

OBPXAInvokeTxRequest invokeTxRequest = new OBPXAInvokeTxRequest(channel,
chaincode, args);
invokeTxRequest.setEndorsers(endorsersArray); // optional blockchain
transaction request attributes
invokeTxRequest.setTransientMap(transientMap); // optional blockchain
transaction request attributes
invokeTxRequest.setTimeout(60000); // optional blockchain transaction
request attributes

obpxaConnection.createXAInvokeTransaction(xid, invokeTxRequest);
```

### Scenario: Explore XA Transactions Using Samples

The following scenario is similar to the one described for atomic transactions: Scenario: Explore Atomic Transactions Using Samples, which uses the Balance Transfer and Marbles samples that are included with Oracle Blockchain Platform.

In this scenario, you install the Balance Transfer and Marbles samples on two different instances of Oracle Blockchain Platform. Each instance then corresponds to an XA data source:

- XA resource OBP-1, with the Marbles chaincode installed on the `goods` channel
- XA resource OBP-2, with the Balance Transfer chaincode installed on the `wallet` channel

In this scenario, you can use an XA transaction that spans multiple data sources to ensure that the exchange of funds and the marble transfer occur in an atomic manner, where either all operations succeed or none succeed. The following code illustrates this scenario:

```
OBPXADataSource obpxaDS1 = ... // create an XA data source, supplying
details about the OBP-1 instance
OBPXADataSource obpxaDS2 = ... // create an XA data source, supplying
details about the OBP-2 instance

// start a global transaction in the client application
// invoke marble transfer on OBP-1
```

```
OBPXAConnection obpxaConn1 = (OBPXAConnection)
obpxaDS1.getXAConnection();
OBPXAInvokeTxRequest invokeMarbleTransferReq = new
OBPXAInvokeTxRequest("goods", "obcs-marbles", new String[]
{"transferMarble", "marble1", "tom"});
obpxaConn1.createXAInvokeTransaction(xid1, invokeMarbleTransferReq);
.
.
.
// invoke fund transfer on OBP-2
OBPXAConnection obpxaConn2 = (OBPXAConnection)
obpxaDS2.getXAConnection();
OBPXAInvokeTxRequest invokeBalanceTransferReq = new
OBPXAInvokeTxRequest("wallet", "obcs-example02", new String[]
{"invoke", "a", "b", "50"});
obpxaConn2.createXAInvokeTransaction(xid2, invokeBalanceTransferReq);
.
.
.
// end the global transaction in the client application
```

There is a known issue with the Marbles sample and the default value of the **Marble Owner** field. For more information, see: Known Issues for Oracle Blockchain Platform.

10

# Work With Databases

This topic contains information to help you understand how to query the state database and how to create and configure a rich history database.

**Topics:**



## Query the State Database

This topic contains information to help you understand how to query the state database where the blockchain ledger's current state data is stored.

### What's the State Database?

The blockchain ledger's current state data is stored in the state database.

When you develop Oracle Blockchain Platform chaincodes, you can extract data from the state database by executing rich queries. Oracle Blockchain Platform supports rich queries by using the SQL rich query syntax and the CouchDB find expressions. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

Hyperledger Fabric doesn't support SQL rich queries. If your Oracle Blockchain Platform network contains Hyperledger Fabric participants, then you need to make sure to do the following:

- If your chaincodes contain SQL rich query syntax, then those chaincodes are installed only on member peers using Oracle Blockchain Platform.
- If a chaincode needs to be installed on Oracle Blockchain Platform and Hyperledger Fabric peers, then use CouchDB syntax in the chaincodes and confirm that the Hyperledger Fabric peers are set up to use CouchDB as their state database repository. Oracle Blockchain Platform can process CouchDB.

**How Does Oracle Blockchain Platform Work with Berkeley DB?**

Oracle Blockchain Platform uses Oracle Berkeley DB as the state database. Oracle Blockchain Platform creates relational tables in Berkeley DB based on the SQLite extension. This architecture provides a robust and performant way to validate SQL rich queries.

For each channel chaincode, Oracle Blockchain Platform creates a Berkeley DB table. This table stores state information data, and contains at least a key column named `key`, and a value column named `value` or `valueJson`, depending on whether you're using JSON format data.

| Column Name | Type | Description |
|---|---|---|
| `key` | TEXT | Key column of the state table. |

| Column Name | Type | Description |
|---|---|---|
| `value` | TEXT | Value column of the state table. |
| `valueJson` | TEXT | JSON format value column of the state table. |

Note that the `valueJson` and `value` columns are mutually-exclusive. So, if the chaincode assigns a JSON value to a key, then the `valueJson` column will hold that value, and the value column will be set to null. If the chaincode assigns a non-JSON value to a key, then the `valueJson` column will be set to null, and the value column will hold the value.

**Example of a State Database**

These are examples of keys and their values from the Car Dealer sample's state database:

| key | value | valueJson |
|---|---|---|
| abg1234 | null | {"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| abg1235 | null | {"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| ser1236 | null | {"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| bra1238 | null | {"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| dtrt10001 | null | {"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979 |

# Rich Queries in the Console

Administrators can run and analyze rich queries from the console.

1. Go to the console and select the **Channels** tab.
2. In the channels table, locate the channel where you want to run a query, click the channels **More Actions** button, and then click **Analyze Rich Queries**. The Analyze Rich Queries dialog box is displayed.
3. To run a rich query against the state database, select **Query Execution**.

a. For **Chaincode**, select the chaincode that is deployed to the channel that you want to query.

b. For **Peer**, select the peer to query.

   Only peers in the current organization that are running the selected chaincode are available.

c. For **Rich Query**, enter the rich query to run and analyze.

   The query format must follow the rich query syntax. For more information about rich query syntax, see Supported Rich Query Syntax.

d. For **Result Rows Limit**, move the slider to the maximum number of result rows to fetch. You can fetch up to 50 rows of results.

4. To get the execution plan for a query, select **Query Plan Explain**. A query execution plan is the sequence of operations that was performed to run the query.

   a. For **Chaincode**, select the chaincode that is deployed to the channel that you want to query.

   b. For **Peer**, select the peer to query.

   c. For **Collection**, select the state database or private data collection.

   d. For **Rich Query**, enter the rich query.

      The `explain` keyword is not needed for this query.

      For example: `select * from <state>`

5. Click **Execute**. The **Results** field shows the query result table or the execution plan. To export the results table as a `.csv` file, click **Export**.

   The results table size is limited to 1 MB. You might need to refine your query to avoid exceeding this limit.

# Supported Rich Query Syntax

Oracle Blockchain Platform supports two types of rich query syntax that you can use to query the state database: SQL rich query and CouchDB rich query.

## SQL Rich Query Syntax

The Berkeley DB JSON extensions are in the form of SQL functions.

**Before You Begin**

Note the following information:

- You can only access the channel chaincode (<STATE>) that you're executing your query from.
- Only the SELECT statement is supported.
- You can't modify the state database table.
- A rich query expression can have only one SELECT statement.
- The examples in this topic are just a few ways that you can write your rich query. You've access to the usual full SQL syntax to query a SQL database.
- You've access to the JSON1 Extension (SQLite extension). See JSON1 Extension and SQL As Understood by SQLite.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

### How to Refer to the State Database in Queries

The state database table name is internally managed by Oracle Blockchain Platform, so you don't need to know the state database's physical name when you write a chaincode.

Instead, you must use the `<STATE>` alias to refer to the table name. For example: `select key, value from <STATE>`.

Note that the `<STATE>` alias is **not** case-sensitive, so you can use either `<state>`, `<STATE>`, or something like `<StAtE>`.

### Retrieve All Keys

Use this syntax:

```
SELECT key FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following list of keys:

**key**

abg1234

abg1235

ser1236

bra1238

dtrt10001

### Retrieve All Keys and Values Ordered Alphabetically by Key

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM  <state> ORDER
BY key
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

| serialNumber | details |
|---|---|
| abg1234 | {"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| abg1235 | {"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |
| bra1238 | {"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |

| serialNumber | details |
|---|---|
| dtrt10001 | {"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979 |
| ser1236 | {"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979} |

**Retrieve All Keys and Values Starting with “abg”**

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM <state> WHERE key LIKE
'abg%'SELECT key, value FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following results:

| serialNumber | details |
|---|---|
| abg1234 | {"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 2020", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"} |
| abg1235 | {"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 4050", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"} |

**Retrieve All Keys with Values Containing a Vehicle Part Owned by "Detroit Auto"**

Use this syntax:

```
SELECT key FROM <state> WHERE json_extract(valueJson, '$.docType') =
'vehiclePart' AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following list of keys:

**key**

abg1234

abg1235

ser1236

bra1238

**Retrieve Model and Manufacturer for all Cars Owned by "Sam Dealer"**

Use this syntax:

```
SELECT json_extract(valueJson, '$.model') AS model, json_extract(valueJson,
'$.manufacturer') AS manufacturer FROM <state> WHERE json_extract(valueJson,
'$.docType') = 'vehicle' AND json_extract(valueJson, '$.owner') = 'Sam Dealer'
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

| model | manufacturer |
|---|---|
| a coupe | Detroit Auto |

If the state value is JSON array, you may use this syntax to retrieve model and manufacturer for all cars owned by "Sam Dealer":

```
SELECT json_extract(j.value, '$.model') AS model, json_extract(j.value,
'$.manufacturer') AS manufacturer FROM <state> s,
json_each(json_extract(s.valueJson,'$')) j WHERE json_valid(j.value) AND
json_extract(j.value, '$.owner') = 'Sam Dealer'
```

## CouchDB Rich Query Syntax

Use the information in this topic if you're migrating your chaincodes containing CouchDB syntax to Oracle Blockchain Platform, or if you need to write chaincodes to install on Hyperledger Fabric peers participating in an Oracle Blockchain Platform network.

If you're writing a new chaincode, then Oracle recommends that you use SQL rich queries to take advantage of the performance benefits that Oracle Blockchain Platform with Berkeley DB provides.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

### Unsupported Query Parameters and Selector Syntax

Oracle Blockchain Platform doesn't support the `use_index` parameter. If used, Oracle Blockchain Platform ignores this parameter, and it will automatically pick the indexes defined on the StateDB in question.

| Parameter | Type | Description |
|---|---|---|
| use_index | json | Instructs a query to use a specific index. |

### Retrieve All Models, Manufacturers, and Owners of Cars, and Order Them by Owner

Use this expression:

```
{
  "fields": ["model", "manufacturer", "owner"],
  "sort": [
    "owner"
   ]
}
```

**Retrieve Model and Manufacturer for All Cars Owned by “Sam Dealer”**

Use this expression:

```
{
  "fields": ["model", "manufacturer"],
  "selector": {
    "docType"  : "vehicle",
     "owner" : "Sam Dealer"
  }
}
```

# State Database Indexes

The state database can contain a large amount of data. In such cases Oracle Blockchain Platform uses indexes to improve data access.

**Default Indexes**

When a chaincode is deployed, Oracle Blockchain Platform creates two indexes.

- Key index — Created on the key column.
- Value index — Created on the value column.

**Custom Indexes**

In some cases, you might need to create custom indexes. You define these indexes using any expression that can be resolved in the context of the state table. Custom indexes created against Berkeley DB rely on the SQLite syntax, but they otherwise follow the same CouchDB implementation provided by Hyperledger Fabric.

Note that you can use custom indexes to dramatically improve the performance of WHERE and ORDER BY statements on large data sets. Because using custom indexes slows down data insertions, you should use them judiciously.

Each custom index is defined as an array of expressions, which support compound indexes, expressed as a JSON document inside one file (note that there's one index per file). You must package this file with the chaincode in a folder named “indexes” in the following directory structure: `statedb/relationaldb/indexes`. See How to add CouchDB indexes during chaincode installation.

**Example Custom Indexes**

The custom index examples in this section use the Car Dealer sample.

**Example 1** —This example indexes the use of the `json_extract` expression in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … ORDER BY json_extract(valueJson, '$.owner')
```

**Example 2** — This example indexes the compound use of the two `json_extract` expressions in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.docType')",
"json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

**Example 3** — This example creates two indexes: the index described in Example 1 and the index described in Example 2. Note that each JSON structure needs to be included in a separate file. Each file describes a single index: a simple index like Example 1, or a compound index like Example 2.

Index 1: `{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}`

Index 2: `{"indexExpressions": ["json_extract(valueJson, '$owner')",
"json_extract(valueJson, '$.docType')"]}`

In the following example, Index 2 is applied to the `AND` expression in the `WHERE` portion of the query, while Index 1 is applied to the `ORDER BY` expression:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto' ORDER BY
json_extract(valueJson, '$.owner')
```

**JSON Document Format**

The JSON document must be in the following format:

```
{"indexExpressions": [expr1, ..., exprN]}
```

For example:

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

## Differences in the Validation of Rich Queries

In some cases, the standard Hyperledger Fabric with CouchDB rich query and the Oracle Berkeley DB rich query behave differently.

In standard Hyperledger Fabric with CouchDB, each key and value pair returned by the query is added to the transaction's read-set and is validated at validation time and without re-executing the query. In Berkeley DB, the returned key and value pair isn't added to the read-set, but the rich query's result is hashed in a Merkle tree and validated against the re-execution of the query at validation time.

Native Hyperledger Fabric doesn't provide data protection for rich query. However, Berkeley DB contains functionality that protects and validates the rich query by adding the Merkle tree hash value into the read-set, re-executing the rich query, and at the validation stage re-calculating the Merkle tree value. Note that because validation is more accurate in Oracle Blockchain Platform with Berkeley DB, chaincode invocations are sometimes flagged for more frequent phantom reads.

# Create the Rich History Database

This topic contains information to help you specify an Oracle database connection and choose channels to create the rich history database. You'll use this database to make analytics reports and visualizations of your ledger's activities.

# What's the Rich History Database?

The rich history database is external to Oracle Blockchain Platform and contains data about the blockchain ledger's transactions on the channels you select. You use this database to create analytics reports and visualization about your ledger's activities.

For example, using the rich history database, you could create analytics to learn the average balance of all of the customers in your bank over some time interval, or how long it took to ship merchandise from a wholesaler to a retailer.

Internally, Oracle Blockchain Platform uses the Hyperledger Fabric history database to manage the ledger and present ledger transaction information to you in the console. Only the chaincodes can access this history database, and you can't expose the Hyperledger Fabric history database as a data source for analytical queries. The rich history database uses an external Oracle database and contains many details about every transaction committed on a channel. This level of data collection makes the rich history database an excellent data source for analytics. For information about the data that the rich history database collects, see Rich History Database Tables and Columns.

You can only use an Oracle database such as Oracle Autonomous Data Warehouse or Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure to create your rich history database. You use the Oracle Blockchain Platform console to provide the connection string and credentials to access and write to the Oracle database. Note that the credentials you provide are the database's credentials and Oracle Blockchain Platform doesn't manage them. After you create the connection, you'll select the channels that contain the ledger data that you want to include in the rich history database. See Enable and Configure the Rich History Database.

You can use standard tables or blockchain tables to store the rich history database. Blockchain tables are tamperproof append-only tables, which can be used as a secure ledger while also being available for transactions and queries with other tables. For more information, see Oracle Blockchain Table.

You can use any analytics tool, such as Oracle Analytics Cloud or Oracle Data Visualization Cloud Service, to access the rich history database and create analytics reports or data visualizations.

# Create the Oracle Database Classic Cloud Service Connection String

You must collect information from the Oracle Database Classic Cloud Service deployed on Oracle Cloud Infrastructure to build the connection string required by the rich history database. You must also enable access to the database through port 1521.

**Find and Record Oracle Database Classic Cloud Service Information**

The information you need to create a connection to the Oracle Database Classic Cloud Service is available in the Oracle Cloud Infrastructure Console.

1. From the Infrastructure Console, click the navigation menu in the top left corner, and then click **Database**.
2. Locate the database that you want to connect to and record the **Public IP** address.
3. Click the name of the database that you want to connect to and record the values in these fields:
   - **Database Unique Name**

- **Host Domain Name**
- **Port**

4. Find a user name and password of a database user with permissions to read from this database, and make a note of these. For example, the user SYSTEM.

**Enable Database Access Through Port 1521**

Add an ingress rule that enables the rich history database to access the database through port 1521.

1. In the Oracle Cloud Infrastructure home page, click the navigation icon and then under **Databases** click **DB Systems**.
2. Click the database that you want to connect to.
3. Click the **Virtual Cloud Network** link.
4. Navigate to the appropriate subnet, and then under **Security Lists**, click **Default Security List For <Target Database>**.

   The Security List page is displayed.
5. Click **Edit All Rules**.
6. Add an ingress rule to allow any incoming traffic from the public internet to reach port 1521 on this database node, with the following settings:
   - **SOURCE CIDR**: 0.0.0.0/0
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 1521
   - **Allows**: TCP traffic for ports: 1521

**Build the Connection String**

After enabling access to the Oracle database, use the information you collected to build the connection string in the Configure Rich History dialog.

Construct the connection string using the following syntax: *<publicIP>*:*<portNumber>*/*<database unique name>*.*<host domain name>*

For example, 192.0.2.0:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.example.com

**Ensure the Database User has Correct Privileges**

In order for the rich history functionality to be able to manage its database sessions and to recover from temporary database or network downtime, ensure the database user registered with Oracle Blockchain Platform has the following two privileges:

```
grant select on v_$session to <user>;
grant alter system to <user>;
```

Additionally, if the rich history database uses Oracle Autonomous Data Warehouse, the database user must have the following privilege:

```
grant unlimited tablespace to <user>;
```

If the database user doesn't have those privileges already, they must be granted by the system database administrator.

Without these privileges Oracle Blockchain Platform can replicate to the database but it cannot recover from situations leading to a damaged database session, which prevents the rich history from catching up with recent transactions for an extended period. Without these privileges on Oracle Autonomous Data Warehouse, no rich history data is saved.

## Enable and Configure the Rich History Database

Use the console to provide database connection information and select the channels with the chaincode ledger data that you want to write to the rich history database. By default channels aren't enabled to write data to the rich history database.

Note the following information:

- Each blockchain network member configures its own rich history database.
- You must use an Oracle database. No other database types are supported.
- Each channel that writes to the rich history database must contain at least one peer node.

1. Enter connection and credential information for the Oracle database that you want to use to store rich history information.
   a. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.

      The Configure Rich History dialog box is displayed.

   b. Enter the user name and password required to access the Oracle database.

   c. In the **Connection String** field, enter the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.

      - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
      - If you're using Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String.
      - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`

   d. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload the required credential wallet ZIP file. This file contains client credentials and is generated from the Oracle autonomous database.

> **Note:**
>
> When you open the Configure Rich History dialog box again after you configure rich history, the wallet file name is not displayed. If you update other settings, you must upload the wallet ZIP file again before clicking **Save**. If you click **Save** while no wallet file name is displayed, the configuration is updated not to use a wallet file.

e. To use blockchain tables to store the rich history database, select **Use Database Blockchain Table**.

   The underlying database must support blockchain tables. For more information, see Oracle Blockchain Table.

   - To specify the number of days to retain tables and rows, select **Basic Configuration**, and then enter the number of days to retain tables and rows. Enter `0` to retain tables or rows permanently. To prevent further changes to the retention values, select **Locked**.
   - To specify table and row retention by using a data definition language (DDL) statement, select **Advanced Configuration Query** and then enter the DDL statement.

f. Click **Save**.

2. Enable rich history on the channels that contain the chaincode data that you want to write to the rich history database.

   a. Go to the console and select the **Channels** tab.

   b. Locate the channel that contains the chaincode data that you want to write to the rich history database. Click its **More Options** button and select **Configure Rich History**.

      The Configure Rich History dialog is displayed.

   c. Click the **Enable Rich History** check box. To store private data collections in the rich history database, enter a list of private data collection names, separated by commas. For more information about private data collections, see What Are Private Data Collections?. To add transaction details to the rich history database, select the details that you want added. Click **Save**.

The rich history database is configured, but tables are not created in the database immediately. When the next relevant transaction or ledger change happens, the tables are created in the rich history database.

## Modify the Connection to the Rich History Database

You can change the rich history database's connection information.

After tables are created in the database for a channel, modifying the rich history configuration for the channel has no effect, even after you click **Save**, unless you change the user name and password or the connection string. If you change the user name and password, tables are created in the same database. If you change the connection string and credentials, a different database is configured, and tables are created after the next relevant transaction or ledger change. You cannot change a rich history database from standard tables to blockchain tables, and you cannot change retention times, unless you also change the credentials or connection string.

1. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.
2. If needed, update the user name and password required to access the Oracle database.
3. If needed, in the **Connection String** field, modify the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.
   - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
   - If you're using Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String.
   - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`
4. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload or re-upload the required credential wallet file. This file contains client credentials and is generated from the Oracle autonomous database.

   > **Note:**
   >
   > When you open the Configure Rich History dialog box again after you configure rich history, the wallet file name is not displayed. If you update other settings, you must upload the wallet ZIP file again before clicking **Save**. If you click **Save** while no wallet file name is displayed, the configuration is updated not to use a wallet file.

5. To use blockchain tables to store the rich history database, select **Use Database Blockchain Table**.

   The underlying database must support blockchain tables.
   - To specify the number of days to retain tables and rows, select **Basic Configuration**, and then enter the number of days to retain tables and rows. Enter `0` to retain tables or rows permanently. To prevent further changes to the retention values, select **Locked**.
   - To specify table and row retention by using a data definition language (DDL) statement, select **Advanced Configuration Query** and then enter the DDL statement.
6. Click **Save**.

## Configure the Channels that Write Data to the Rich History Database

You can enable channels to write chaincode ledger data to the rich history database, and you can stop channels from writing data to the rich history database. You can also configure an

individual channel to use a different rich history database configuration than the global setting.

You must specify the global information to connect to the rich history database before you can select channels that write to the rich history database. See Enable and Configure the Rich History Database.
After tables are created in the database for a channel, modifying the rich history configuration for the channel has no effect, even after you click **Save**, unless you change the user name and password or the connection string. If you change the user name and password, tables are created in the same database. If you change the connection string and credentials, a different database is configured, and tables are created after the next relevant transaction or ledger change. You cannot change a rich history database from standard tables to blockchain tables, and you cannot change retention times, unless you also change the credentials or connection string.

1. Go to the console and select the **Channels** tab.
2. Locate the channel that you want to modify access for. Click its **More Options** button and select **Configure Rich History**.

   The Configure Rich History dialog is displayed.
3. To enable collection of rich history data for the channel, select the **Enable Rich History** check box. To disable collection of rich history data for the channel, clear the **Enable Rich History** check box.
4. To configure the channel to collect rich history data using a different database or different settings, select **Use channel level configuration**, and then specify the settings to use.

   For more information about the rich history settings, see Enable and Configure the Rich History Database.
5. Click **Save**.

## Monitor the Rich History Status

After configuring the rich history database, you can use the console to monitor the rich history replication status.

1. Go to the console and select the **Channels** tab.
2. In the channels table, click the **More Actions** button for the channel that you want to monitor, and then click **Rich History Status**.

   The Rich History Status dialog box is displayed, which includes details about replication and configuration status.
3. Click **Refresh** to display the latest status.

## Limit Access to Rich History

You can use channel policies and access control lists (ACLs) to limit the organizations that can configure the rich history database and retrieve rich history status or configuration information.

By default, all organizations that have administrative access to a channel can configure rich history collection and can retrieve rich history status and configuration details. To limit this access to, for example, the founder organization, you create a channel policy and apply the policy to the resources that control access.

1. Go to the console and select the **Channels** tab.

   The **Channels** tab is displayed. The channel table contains a list of all of the channels on your network.

2. In the channel table, click the name of the channel where you want to limit access.

3. Click **Channel Policies**, and then create a signature policy that includes the organization members that will access the rich history functions.

   For more information about channel policies, see Work With Channel Policies and ACLs.

   For example, create a policy that includes only the identity of the founder organization, not the identity of any participant organizations.

4. Click **ACLs**.

5. In the Resources table, locate the resource that you want to update to use the new policy. Click **Expand** for the resource and then select the policy to assign to the resource

   The following table shows the resources that control access to rich history.

| Resource | Access control |
|---|---|
| `obpadmin/ ConfigureRichHistoryChannel` | Controls configuring, enabling, and disabling rich history for a channel. |
| `obpadmin/ GetRichHistoryChannelStatus` | Controls retrieving rich history replication status for a channel. |
| `obpadmin/ GetRichHistoryChannelConfig` | Controls retrieving the current rich history configuration for a channel. |

6. Click **Update ACLs**.

The rich history access is now controlled by the new policy. Organization members that are not included in the new policy will receive an error message when they attempt to access a resource that is controlled by the policy.

## Rich History Database Tables and Columns

The rich history database contains three tables for each channel: history, state, and latest height. You'll query the history and state tables when you create analytics about your chaincodes' ledger transactions. If you've chosen to select any of the transaction details when enabling the rich history, an additional table will be created with the transaction details.

**History Table**

The <*instanceName*><*channelName*>_hist table contains ledger history. The data in this table tells you the chaincode ID, key used, if the transaction was valid, the value assigned to the key, and so on.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

If configured, private data is also stored in this table. For private data, the chaincode ID uses the following format: <*chaincodeName*>$$<*collectionName*>.

| Column | Datatype |
|---|---|
| chaincodeId | VARCHAR2 (256) |
| key | VARCHAR2 (1024) |
| txnIsValid | NUMBER (1) |
| value | VARCHAR2 (4000) |
| valueJson | CLOB |
| blockNo | NUMBER NOT NULL |
| txnNo NUMBER | NOT NULL |
| txnId | VARCHAR2 (128) |
| txnTimestamp | TIMESTAMP |
| txnIsDelete | NUMBER (1) |

### State Table

The *<instanceName><channelName>*_state table contains data values replicated from the state database. You'll query the state table when you create analytics about the state of the ledger.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

| Column | Datatype |
|---|---|
| chaincodeId | VARCHAR2 (256) |
| key | VARCHAR2 (1024) |
| value | VARCHAR2 (4000) |
| valueJson | CLOB |
| blockNo | NUMBER |
| txnNo | NUMBER |

### Latest Height Table

The *<instanceName><channelName>*_last table is used internally by Oracle Blockchain Platform to track the block height recorded in the rich history database. It determines how current the rich history database is and if all of the chaincode transactions were recorded in the rich history database. You can't query this database for analytics.

### Transaction Details Table

The *<instanceName><channelName>*_more table contains attributes related to committed transactions. When enabling the rich history database, you can select which of these attributes you want to record in this table. The transaction details table only captures information about endorser transactions - not configuration transactions or any other kind of Hyperledger Fabric transactions.

| Column | Datatype |
|---|---|
| CHAINCODEID | VARCHAR2 (256) |

| Column | Datatype |
|---|---|
| BLOCKNO | NUMBER |
| TXNNO | NUMBER |
| TXNID | VARCHAR2(128) |
| TXNTIMESTAMP | TIMESTAMP |
| SUBMITTERCN | VARCHAR2(512) |
| SUBMITTERORG | VARCHAR2(512) |
| SUBMITTEROU | VARCHAR2(512) |
| CHAINCODETYPE | VARCHAR2(32) |
| VALIDATIONCODENAME | VARCHAR2(32) |
| ENDORSEMENTS | CLOB |
| INPUTS | CLOB |
| EVENTS | CLOB |
| RESPONSESTATUS | NUMBER(0) |
| RESPONSEPAYLOAD | VARCHAR2(1024) |
| RWSET | CLOB |
| BLOCKCREATORCN | VARCHAR2(512) |
| BLOCKCREATORORG | VARCHAR2(512) |
| BLOCKCREATOROU | VARCHAR2(512) |
| CONFIGBLOCKNUMBER | NUMBER(0) |
| CONFIGBLOCKCREATORCN | VARCHAR2(512) |
| CONFIGBLOCKCREATORORG | VARCHAR2(512) |
| CONFIGBLOCKCREATOROU | VARCHAR2(512) |

**Note:**

- Organization (ORG) and organization unit (OU) are driven by identity certificates, which implies that they may be assigned to multiple values. They are captured as a comma separated list in the table's values.
- For identities, the table includes information only about the "Subject" portion of the certificates, not the "Issuer" one.
- The `RWSET` column contains operations on all chaincodes (in the same ledger) performed during endorsement. As such, you will typically see both lscc read operations and the actual chaincode namespace operations.

A

# Node Configuration

This topic contains information to help you understand and configure your nodes. Each node type has different configuration options.

**Topics:**



## CA Node Attributes

A certificate authority (CA) node keeps track of identities and certificates on the blockchain network.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

**Table A-1 CA Node Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Fabric CA ID | This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can't modify this ID. | ca |
| Listen Port | This is the listening port that Oracle Blockchain Platform assigned to the node. You can't change the port number. | Specific to your organization. |
| Max Enrollments | Use this field to determine how many times the CA server allows a password to be used for enrollment on the network. Consider the following options:<br>• -1 — The server allows a password to be used an unlimited number of times for enrollment. | -1 |
| Log Level | Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use INFO. | INFO |

# Console Node Attributes

The console node manages the performance of the console.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

**Table A-2 Console Node Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Console ID | This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. | console |
| Local MSP ID | This is the assigned MSP ID for your organization. You can't modify this ID. | NA |
| Listen Port | This is the listening port that Oracle Blockchain Platform assigned to the node. You can't change the port number. | Specific to your organization. |
| Log Level | Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR. | INFO |
| Request Timeout (s) | Specify the maximum amount of time in seconds that you want the console to attempt to contact the nodes before timing out. | 600 |

# Orderer Node Attributes

An orderer node collects transactions from peer nodes, bundles them, and submits them to the blockchain ledger. The node's attributes determine how the node performs and behaves on the network.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

**Table A-3 Orderer Node — General Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Orderer ID | This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. | orderer<number-partition> |
| Local MSP ID | This is the assigned MSP ID for your organization. You can't modify this ID. | NA |

**Table A-3 (Cont.) Orderer Node — General Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Listen Port | This is the listening port that Oracle Blockchain Platform assigned to the node. You can't change the port number. | Specific to your organization. |
| Log Level | Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR. | INFO |

**Table A-4 Orderer Node — Advanced Attributes — Raft/Cluster tab**

| Attribute | Description | Default Value |
|---|---|---|
| SendBufferSize | The maximum number of messages in the egress buffer. Consensus messages are dropped if the buffer is full, and the transaction messages are waiting for space to be freed. | 10 |
| DialTimeout in seconds | The maximum duration of time after which connection attempts are considered as failed. | 5 |
| RPCTimeout in seconds | The maximum duration of time after which RPC attempts are considered as failed. | 7 |
| Replication/BufferSize in bytes | The maximum number of bytes that can be allocated for each in-memory buffer used for block replication from other cluster nodes. | 20971520 |
| Replication/ BackgroundRefreshInterval in minutes | The time between two consecutive attempts to replicate existing channels that this node was added to, or channels that this node failed to replicate in the past. | 5 |
| Replication/RetryTimeout in seconds | The maximum duration the ordering node will wait between two consecutive attempts. | 5 |
| Replication/PullTimeout in seconds | The maximum duration the ordering node will wait for a block to be received before it aborts. | 5 |
| Consensus/EvictionSuspicion in minutes | The threshold that a node will start suspecting its own eviction if it has been leaderless for this period of time. | 2 |

# Peer Node Attributes

A peer node reads, endorses, and writes transactions to the blockchain ledger. The node's attributes determine how the node performs and behaves on the network.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

**Table A-5 Peer Node — General Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Peer ID | This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. | peer0 |
| Local MSP ID | This is the assigned MSP ID for your organization. You can't modify this ID. | Specific to your organization. |
| Role | Specifies if the peer's role is Member or Admin. In most cases this field displays Member.<br>This role is used by the chaincode's endorsement policy. The endorsement policy specifies the MSP that must validate the identity of the signer peer and the signer peer's role. The Admin role is normally assigned in situations where you want to further protect sensitive operations and make sure that those operations are endorsed by specific peers.<br>The peers created with your instance were assigned the Member role. | Member |
| Listen Port | This is the listening port that Oracle Blockchain Platform assigned to the node. You can't change the port number. | Specific to your organization. |
| Log Level | Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR. | INFO |
| Alias | Optionally, enter text to further identify the peer beyond the peer ID. | NA |

**Table A-6 Peer Node — Advanced Attributes — Gossip tab**

| Attribute | Description | Default Value |
|---|---|---|
| Bootstrap Peers | Provide the service name address and port that the peer uses to contact other peers during startup. This endpoint must match the endpoints of the peers in the same organization. | NA |
| Max Block Count to Store | Enter the maximum number of blocks to store in memory. | 10 |
| Max Propagation Burst Latency in milliseconds | Enter how many milliseconds between message pushes. | 10 |
| Max Propagation Burst Size | Enter the number of messages to be stored until a push remote peer is triggered. | 10 |
| Propagate Iterations | Enter the number of times a message is pushed to the peers. | 1 |
| Max Connection Attempts | Enter the maximum number of attempts to make when connecting to a peer. | 120 |
| Message Expiration Factor | Enter the message expiration factor for alive messages. | 20 |
| Propagate Peer Number | Enter how many peers to send messages to. | 3 |
| Pull Interval in seconds | Enter how many seconds between pull phases. | 4 |
| Pull Peer Number | Enter the number of peers to pull from. | 3 |
| Request State Info Interval in seconds | Enter how often to pull state information messages from the peers. | 4 |
| Publish State Info Interval in seconds | Enter how often to send state information messages to the peers. | 4 |
| Publish Cert Period in seconds | Enter how many seconds from startup that certificates are included in alive messages. | 10 |
| Dial Timeout in seconds | Enter how many seconds before dial times out. | 3 |
| Connect Timeout in seconds | Enter how many seconds until the connection times out. | 2 |
| Receive Buffer Size | Enter the size of the buffer for received messages. | 20 |
| Send Buffer Size | Enter the size of the buffer for sending messages. | 200 |
| Digest Wait Time in seconds | Enter how many seconds to wait before the pull engine processes incoming digests. | 1 |
| Request Wait Time in seconds | Enter how many seconds to wait before the pull engine removes incoming nonce. | 1,500 |

**Table A-6 (Cont.) Peer Node — Advanced Attributes — Gossip tab**

| Attribute | Description | Default Value |
|---|---|---|
| Response Wait Time in seconds | Enter how many seconds that the pull engine waits before it terminates the pull. | 2 |
| Alive Time Interval in seconds | Enter how often to check alive time. | 5 |
| Alive Expiration Timeout in seconds | Enter how many seconds to wait before the alive expiration times out. | 25 |
| Reconnect Interval in seconds | Enter how many seconds to wait before reconnecting. | 25 |
| Skip Block Verification | Click to skip block verification. | Not selected |

**Table A-7 Peer Node — Advanced Attributes — Gossip/Election tab**

| Attribute | Description | Default Value |
|---|---|---|
| Membership Sample Interval in seconds | How often in seconds the peer checks its stability on the network. | 1 |
| Leader Alive Threshold in seconds | The number of seconds to elapse before the last declaration message is sent and before the peer determines leader election. | 10 |
| Leader Election Duration in seconds | The number of seconds to elapse after the peer sends the propose message and declares itself leader. | 5 |

**Table A-7 (Cont.) Peer Node — Advanced Attributes — Gossip/Election tab**

| Attribute | Description | Default Value |
|---|---|---|
| Leader | A channel's leader peer receives blocks and distributes them to the other peers within the cluster. Specify the mode that you want the peer to use to determine a leader.<br>• **OrgLeader** — Select this option to use static leader mode and make the peer the organization leader. If you select this option and then add more peers to the channel, then you must set all peers to **OrgLeader**.<br>• **UseLeaderElection** — Select this option to use dynamic leader election on the channel. Before an active leader is selected for the organization, the system must run the configuration transaction to add the organization to the channel, and then the system updates the new peers with the configuration transaction. | OrgLeader |

**Table A-8 Peer Node — Advanced Attributes — Event Service tab**

| Attribute | Description | Default Value |
|---|---|---|
| Buffer Size | Enter the maximum number of events that the buffer can contain. The system won't send the events that exceed this number. | 100 |
| Timeout in milliseconds | Enter in milliseconds the maximum time allowed for the business network to send an event. | 10 |

**Table A-9 Peer Node — Advanced Attributes — Chaincode tab**

| Attribute | Description | Default Value |
|---|---|---|
| Startup timeout in seconds | Enter in seconds the maximum time to wait between when the container starts and the registry responds. | 300 |

**Table A-9 (Cont.) Peer Node — Advanced Attributes — Chaincode tab**

| Attribute | Description | Default Value |
|---|---|---|
| Execute timeout in seconds | Enter in seconds the maximum time that a chaincode attempts to execute before timing out. | 30 |
| Mode | Displays how the system runs the chaincode. This value is always net. | net |
| Keepalive in seconds | If you're using a proxy for communication, then enter in seconds the maximum amount of time to keep the connection between a peer and the chaincode alive. | 0 |
| Log Level | Specify the log level that you want to use for all loggers in the chaincode container. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR. | INFO |
| Shim Level | Specify the log level that you want to use for the shim logger. | WARNING |

# REST Proxy Node Attributes

A REST proxy node allows you to query or invoke a chaincode through the RESTful protocol. The node's attributes determine how the node performs on the network and which channel, chaincode, and peers are used in the transactions performed by the node.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

**Table A-10 REST Proxy Node Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| REST Proxy Name | This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can't modify this ID. | restproxy |
| Proposal Wait Time (ms) | Enter the number of milliseconds that the node waits for completion of the proposal process. If this number is exceeded, then the transaction times out. | 60,000 |

**Table A-10 (Cont.) REST Proxy Node Attributes**

| Attribute | Description | Default Value |
|---|---|---|
| Transaction Wait Time (ms) | Enter the number of milliseconds that the node waits for execution after the transaction is submitted. If this number is exceeded, then the transaction times out. | 300,000 |
| Log Level | Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use WARNING or ERROR. | INFO |
| Transaction Event Logging | If you specify a log level of INFO, you can also enable or disable transactional event logging. Transaction events are not logged when the log level is WARNING or ERROR, and they are always logged when the log level is DEBUG. | Disabled |

B

# Using the Fine-Grained Access Control Library Included in the Marbles Sample

Starting in v1.2, Hyperledger Fabric provided fine-grained access control to many of the management functions. Oracle Blockchain Platform provides a marbles sample package on the Developer Tools tab of the console, implementing a library of functions that chaincode developers can use to create access control lists for chaincode functions. It currently only supports the Go language.

### Background

The goal of this sample access control library is to provide the following:

- Provides a mechanism to allow you to control which users can access particular chaincode functions.
- The list of users and their entitlements should be dynamic and shared across chaincodes.
- Provides access control checks so that a chaincode can check the access control list easily.
- At chaincode deployment time, allows you to populate the list of resources and access control lists with your initial members.
- An access control list must be provided to authorize users to perform access control list operations.

### Download the Sample

On the **Developer Tools** tab, open the **Samples** pane. Click the download link under **Marbles with Fine-Grained ACLs**. This package contains three sub-packages:

- `Fine-GrainedAccessControlLibrary.zip`:
  The fine-grained access control library. It contains functions in Go which can be used by chaincode developers to create access control lists for chaincode functions.
- `fgACL_MarbleSampleCC.zip`:
  The marbles sample with access control lists implemented. It includes a variety of functions to let you examine how to work with fine-grained access control lists, groups and resources to restrict functions to certain users/identities.
- `fgACL-NodeJSCode.zip`:
  Node.js scripts which use the Node.js SDK to run the sample. `registerEnrollUser.js` can be used to register new users with the Blockchain Platform. `invokeQueryCC.js` can be used to run transactions against a Blockchain Platform instance.

## Terminology and Acronyms

| Term | Description |
|---|---|
| Identity | An X509 certificate representing the identity of either the caller or the specific identity the chaincode wants to check. |
| Identity Pattern | A pattern that matches one or more identities. The following patterns are suggested:<br>• X.509 Subject Common Name – CN<br>• X.509 Subject Organizational Unit – OU<br>• X.509 Subject Organization – O<br>• Group as defined in this library – GRP<br>• Attribute – ATTR<br>The format for a pattern is essentially just a string with a prefix. For example, to define a pattern that matches any identity in organization "example.com", the pattern would be "%O%example.com". |
| Resource | The name of anything the chaincode wants to control access to. To this library it is just a named arbitrary string contained in a flat namespace. The semantics of the name are completely up to the chaincode. |
| Group | A group of identity patterns. |
| ACL | Access Control List: a named entity that has a list of identity patterns, a list of types of access such as "READ", "CREATE", "INVOKE", "FORWARD", or anything the chaincode wants to use. This library will use access types of CREATE, READ, UPDATE, and DELETE (standard CRUD operations) to maintain its information. Other than those four as they relate to the items in this library, they are just strings with no implied semantics. An application may decide to use accesses of "A", "B", and "CUSTOM". |

# Fine-Grained Access Control Library Functions

The library package provides the following functions for Resources, Groups and ACLs as well as global functions.

### Global Functions

| Function | Description |
|---|---|
| Initialization(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (error) (error) | When the chaincode is instantiated, the Initialization function is called. That function will initialize the world state with some built in access control lists. These built in lists are used to bootstrap the environment. So there needs to be access control on who can create resources, groups, and ACLs. If the identify is nil, then use the caller's identify.<br>After the bootstrap is done, the following entities are created:<br>• A resource named ".Resources". A corresponding ACL named ".Resources.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access.<br>• A group named ".Groups". A corresponding ACL named ".Groups.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access.<br>• An ACL named ".ACLs". A corresponding ACL control list named ".ACLs.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access. |
| NewGroupManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*GroupManager, error) | Get the group manager that's used for all group related operations.<br>Identity: the default identity for related operation. If it's nil, then use caller's identity. |
| NewACLManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ACLManager, error) | Get the ACL manager that's used for all ACL related operations.<br>Identity: the default identity for related operation. If it's nil, then use caller's identity. |
| NewResourceManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ResourceManager, error) | Get the resource manager that's used for all resource related operations.<br>Identity: the default identity for related operation. If it's nil, then use caller's identity. |

### Access Control List (ACL) Functions

Definition of `ACL` structure:

```
type ACL struct {
  Name string
  Description string
  Accesses []string  // CREATE, READ, UPDATE, and DELETE, or whatever
the end-user defined
  Patterns []string    // identities
  Allowed bool          // true means allows access.
  BindACLs []string  // The list of ACL , control who can call the
APIs of this struct
}
```

- **Accesses:** The Accesses string is a list of comma-separated arbitrary access names and completely up to the application except for four: CREATE, READ, UPDATE, and DELETE. These access values are used in maintaining the fine grained access control. Applications can use their own access strings such as `"register"`, `"invoke"`, or `"query"`, or even such things as access to field names such as `"owner"`, `"quantity"`, and so on.
- **Allowed:** Allowed determines whether identities that match a pattern are allowed access (true) or prohibited access (false). You could have an access control list that indicates Bob has access to `"CREATE"`, and another one that indicates group Oracle (of which Bob is a member) is prohibited from `"CREATE"`. Whether Bob has access or not depends upon the order of the access control lists associated with the entity in question.
- **BindACLs:** The BindACLs parameter will form the initial access control list.

ACL functions:

| Function | Description |
|---|---|
| Create(acl ACL, identity *x509.Certificate) (error) | Creates a new ACL. Duplicate named ACL are not allowed.<br>To create a new ACL, the identity needs to have CREATE access to the bootstrap resource named ".ACLs". If identity is nil, the default identity specified in newACLManager() is used. |
| Get(aclName string, identity *x509.Certificate) (ACL, error) | Get a named ACL.<br>The identity must have READ access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used. |
| Delete(aclName string, identity *x509.Certificate) (error) | Delete a specified ACL.<br>The identity must have DELETE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used. |

| Function | Description |
|---|---|
| Update(acl ACL, identity *x509.Certificate) (error) | Update an ACL.<br>The identity must have UPDATE access to the named resource, and the named ACL must exist. If identity is nil, the default identity specified in NewACLManager() is used. |
| AddPattern(aclName string, pattern string, identity *x509.Certificate) (error) | Adds a new identity pattern to the named ACL. The identity must have UPDATE access to the named ACL.<br>If identity is nil, the default identity specified in newACLManager() is used. |
| RemovePattern(aclName string, pattern string, identity *X509Certificate) (error) | Removes the identity pattern from the ACL. The identity must have UPDATE access to the named ACL.<br>If identity is nil, the default identity specified in newACLManager() is used. |
| AddAccess(aclname string, access string, identity *X509Certificate) (error) | Adds a new access to the named ACL. The identity must have UPDATE access to the named ACL.<br>If identity is nil, the default identity specified in newACLManager() is used. |
| RemoveAccess(aclName string, access string, identity *X509Certificate) (error) | Removes the access from the ACL. The identity must have UPDATE access to the named ACL.<br>If identity is nil, the default identity specified in newACLManager() is used. |
| UpdateDescription(aclName string, newDescription string, identity *X509Certificate) (error) | Update the description.<br>The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used. |
| AddBeforeACL(aclName string, beforeName string, newBindACL string, identity *X509Certificate) (error) | Adds a bind ACL before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the bind ACL list.<br>The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used. |
| AddAfterACL(aclName string, afterName string, newBindACL string, identity *X509Certificate) (error) | Adds a bind ACL after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the bind ACL list.<br>The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used. |
| RemoveBindACL(aclName string, removeName string, identity *X509Certificate) (error) | Removes the removeName ACL from the bind ACL list.<br>The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used. |

| Function | Description |
|---|---|
| GetAll(identity *x509.Certificate) ([]ACL, error) | Get all the ACLs.<br>The identity must have READ access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used. |

### Group Functions

Definition of `Group` structure:

```
type Group struct {
    Name string
    Description string
    Members []string     // identity patterns, except GRP.
    BindACLs []string    // The list of ACLs, controls who can access
this group.
}
```

Definition of `GroupManager` functions:

| Function | Description |
|---|---|
| Create(group Group, identity *x509.Certificate) (error) | Create a new group.<br>The identity must have CREATE access to bootstrap group ".Group". If identity is nil, the default identity specified in NewGroupManager() is used. |
| Get(groupName string, identity *x509.Certificate) (Group, error) | Get a specified group.<br>The identity must have READ access to this group. If identity is nil, the default identity specified in NewGroupManager() is used. |
| Delete(groupName string, identity *x509.Certificate) (error) | Delete a specified group.<br>The identity must have DELETE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used. |
| AddMembers(groupName string, member []string, identity *x509.Certificate) (error) | Add one or more members into the group.<br>The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used. |
| RemoveMembers(groupName string, member []string, identity *x509.Certificate) (error) | Remove one or more member from a group.<br>The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used. |
| UpdateDescription(groupName string, newDes string, identity *x509.Certificate) (error) | Update the description.<br>The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used. |

| Function | Description |
|---|---|
| AddBeforeACL(groupName string, beforeName string, aclName string, identity *x509.Certificate) (error) | Adds an bind ACL to the group before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource.<br>The identity must have UPDATE access to the named group. If identity is nil, the default identity specified in NewGroupManager () is used. |
| AddAfterACL(groupName string, afterName string, aclName string, identity *x509.Certificate) (error) | Adds a bind ACL to the group after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the group.<br>The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used. |
| RemoveBindACL(groupName string, aclName string, identity *x509.Certificate) (error) | Removes the named ACL from the bind ACL list of the named group.<br>The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used. |
| GetAll(identity *x509.Certificate) ([]Group, error) | Get all groups.<br>The identity must have READ access to these groups. If identity is nil, the default identity specified in NewGroupManager () is used. |

### Resource Functions

Definition of `Resource` structure:

```
type Resource struct {
     Name string
     Description string
     BindACLs []string      // The name list of ACL, controls who can access
this resource
}
```

Resource Functions:

| Fuction | Description |
|---|---|
| Create(resource Resource, identity *x509.Certificate) (error) | Create a new resource. Duplicate named resources are not allowed.<br>The identity needs to have CREATE access to the bootstrap resource named ".Resources" If identity is null, the default identity specified in NewResourceManager() is used. |

| Fuction | Description |
|---|---|
| Get(resName string, identity *x509.Certificate) (Resource, error) | Get a specified resource.<br>The identity must have READ access to the resource. If identity is null, the default identity specified in NewResourceManager() is used. |
| Delete(resName string, identity *x509.Certificate) (error) | Delete a named resource.<br>The identity must have DELETE access to the named resource. If identity is null, the default identity specified in NewResourceManager() is used. |
| UpdateDescription(resourceName string, newDes string, identity *x509.Certificate) (error) | Update the description.<br>The identity must have UPDATE access to this resource. If identity is nil, the default identity specified in NewResourceManager() is used. |
| AddBeforeACL(resourceName string, beforeName string, aclName string, identity *x509.Certificate) (error) | Adds a bind ACL to the resource before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource.<br>The identity must have UPDATE access to the named resource. If identity is nil, the default identity specified in NewResourceManager() is used. |
| AddAfterACL(resourceName string, afterName string, aclName string, identity *x509.Certificate) (error) | Adds a bind ACL to the resource after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the resource.<br>The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used. |
| RemoveBindACL(resourceName string, aclName string, identity *x509.Certificate) (error) | Removes the named ACL from the bind ACL list of the named resource.<br>The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used. |
| CheckAccess(resName string, access string, identity *x509.Certificate) (bool, error) | Check whether the current user has the specified access to the named resource.<br>If the identity is nil, the default identity specified in NewResourceManager() is used. |
| GetAll(identity *x509.Certificate) ([]Resource, error) | Get all resources.<br>The identity must have READ access to these resources. If identity is nil, the default identity specified in NewResourceManager() is used. |

# Example Walkthough Using the Fine-Grained Access Control Library

This topic provides some examples of how this library and chaincode can be used. These all assuming `Init()` has been called to create the bootstrap entities and the

caller of `Init()` and `invoke()` is `"%CN%frank.thomas@example.com"`. The normal flow in an application will be to create some initial access control lists that will be used to grant or deny access to the other entities.

### Initialization

Call `Initialization()` to create bootstrap entities when instantiating chaincode. For example:

```
import "chaincodeACL"
func (t \*SimpleChaincode) Init(nil, stub shim.ChaincodeStubInterface)
pb.Response
{
         err := chaincodeACL.Initialization(stub)
}
```

### Create a new ACL

```
import "chaincodeACL"
...
{

**ACLMgr**  := chaincodeACL.NewACLManager(nil, stub) // Not specify
identity, use caller's identity as default.

// Define a new ACL
**newACL**  := chaincodeACL.ACL{

    "AllowAdmins",   // ACL name
    "Allow admins full access",  // Description
    []string{"CREATE","READ","UPDATE","DELETE"},    // Accesses allowed or
not
    true, // Allowed
    []string{"%CN%bob.dole@example.com","%OU%example.com,"%GRP%admins"}, //
Initial identity patterns
    ".ACLs.acl", // Start with bootstrap ACL

}

// Add this ACL with default identity (caller's identify here)
err :=  **ACLMgr**.Create( **newACL** , nil)

}
```

Now that we have a new ACL, we can use that to modify who can perform certain operations. So we'll first add this new ACL to the bootstrap group `.Groups` to allow any admin to create a group.

### Add an ACL to a group

```
import "chaincodeACL"
…
{
```

```
  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub) // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.AddAfterACL(

    ".Groups",     // Bootstrap group name
    ".Groups.ACL", // Which ACL to add after
    "AllowAdmins", // The new ACL to add
    nil            // with default identity that's frank.thomas

)

}
```

This adds the `AllowAdmins` ACL to the bootstrap group `.Groups` after the initial bootstrap ACL. Thus this ensures that Frank Thomas can still perform operations on `.Groups` as the ACL granting him permission is first in the list. But now anyone that matches the `AllowAdmins` ACL can perform CREATE, READ, UPDATE, or DELETE operations (they can now create new groups).

### Create a new group

Admins can now create a new group:

```
import "chaincodeACL"
...
{

...
  // Define a new group.
  **newGroup**  := chaincodeACL.Group{

      "AdminGrp",   // Name of the group
      "Administrators of the app",   // Description of the group

{"%CN%jill.muller@example.com","%CN%ivan.novak@example.com","%ATTR%role
=admin"},
      []string{"AllowAdmins"},   // The ACL for the group

    }

  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub)   // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.Create( **newGroup** ,
bob\_garcia\_certificate)   // Using a specific certificate

...
}
```

This call is using an explicit identity - that of Bob Garcia (using his certificate) - to try and create a new group. Since Bob Garcia matches a pattern in the `AllowAdmins` ACL and members of that ACL can perform CREATE operations on the bootstrap group `.Groups`, this call will succeed. Had Jim Silva - who was not in organization unit `example.com` nor in the group `AdminGrp` (which still doesn't exist) - had his certificate passed as the last argument, the call would fail as he doesn't have the appropriate

permissions. This call will create a new group called "AdminGrp" with initial members of the group being jill.muller@example.com and ivan.novak@example.com or anyone with the attribute (ABAC) role=admin.

### Create a new resource

```
import "chaincodeACL"
...
{

  ...
  **newResource**  :=  **chaincodeACL**.Resource{

      "transferMarble", // Name of resource to create

      "The transferMarble chaincode function", // Description of the resource

      []string{"AllowAdmins"}, // Single ACL for now allowing admins

  }

  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil, stub)  //
Not specify identity, use caller's identity as default.
  err :=  **resourceMgr**.Create(resourceMgr, nil)   // Using caller's
certificate

  ...
}
```

This would create a new resource named `transferMarble` that the application might use to control access to the `transferMarble` chaincode function. The access is currently limited by the `AllowAdmins` access control list.

### Check access for a resource

We can use this new resource in our chaincode to only allow admins to transfer a marble by modifying the `invoke()` method of the Marbles chaincode as follows:

```
import "chaincodeACL"
…
func (t \*SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {

  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil, stub)   //
Not specify identity, use caller's identity as default.

  function, args := stub.GetFunctionAndParameters()

  fmt.Println("invoke is running " + function)        // Handle different
functions

  if function == "initMarble" {   //create a new marble

      return t.initMarble(stub, args)}
```

```
  else if function == " **transferMarble**" { //change owner of a
specific marble

    **allowed** , err : =  **resourceMgr**. **CheckAccess**
("transferMarble", "UPDATE", nil)
    if  **allowed**  == true {

      return t.transferMarble(stub, args)

    else {

      return NOACCESS

    }

    } else if function == "transferMarblesBasedOnColor" { //transfer
all marbles of a certain color
    …

    }

}
```

# Fine-Grained Access Control Marbles Sample

The marbles chaincode application lets you create assets (marbles) with unique attributes (name, size, color and owner) and trade these assets with fellow participants in a blockchain network.

This sample application includes a variety of functions to let you examine how to work with access control lists and groups to restrict functions to certain users.



### Overview of the Sample

The test scenario included in the sample contains the following restrictions in order to manage assets:

- Bulk transfer of red marbles is only allowed by identities having the `"redMarblesTransferPermission"` Fabric attribute.
- Bulk transfer of blue marbles is only allowed by identities having the `"blueMarblesTransferPermission"` Fabric attribute.
- Deletion of marbles is only allowed to identities with `"deleteMarblePermission"` Fabric attribute.

These restrictions are enforced by implementing the following library methods in the `fgMarbles_chaincode.go` chaincode:

- Create a fine-grained ACL group named `bulkMarblesTransferGroup`. This group will define all the identities which can transfer marbles based on color (bulk transfers):

```
createGroup(stub, []string{" bulkMarblesTransferGroup",
"List of Identities allowed to Transfer Marbles in Bulk",
"%ATTR%redMarblesTransferPermission=true,
%ATTR%blueMarblesTransferPermission=true", ".ACLs"})
```

- Create a fine-grained ACL named `redMarblesAcl` which provides bulk transfer of red marbles access to `bulkMarblesTransferGroup`:

```
createACL(stub, []string{"redMarblesAcl",
"ACL to control who can transfer red marbles in bulk",
"redMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup", "true",
".ACLs"})
```

- Create a fine-grained ACL named `blueMarblesAcl` which provides bulk transfer of blue marbles access to `bulkMarblesTransferGroup`:

```
createACL(stub, []string{"blueMarblesAcl",
"ACL to control who can transfer blue marbles in bulk",
"blueMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup", "true",
".ACLs"})
```

- Create a fine-grained ACL named `deleteMarbleAcl` to restrict marble deletion based on `"canDeleteMarble=true"` Fabric attribute:

```
createACL(stub, []string{"deleteMarbleAcl",
"ACL to control who can Delete a Marble",
"deleteMarblePermission", "%ATTR%deleteMarblePermission=true", "true",
".ACLs"})
```

- Create a fine-grained ACL resource named `marble`, operations on which are controlled using the various ACLs we created:

```
createResource(stub, []string{"marble",
"System marble resource",
"deleteMarbleAcl,blueMarblesAcl,redMarblesAcl,.ACLs"})
```

### Pre-requisites and Setup

In order to run the fine-grained access control version of the marbles sample, complete these steps:

1. Download the fine-grained access control version of the marbles sample. On the **Developer Tools** tab, open the **Samples** pane, and then click the download link under **Marbles with Fine-Grained ACLs**. Extract this package - it contains ZIP files of the marbles sample (`fgACL_MarbleSampleCC.zip`), Node.js files to run the sample (`fgACL-NodeJSCode.zip`), and the fine-grained access control library (`Fine-GrainedAccessControlLibrary.zip`).

2. Hyperledger Fabric v2.x only: Generate the chaincode package that will be deployed to Blockchain Platform:
   - Extract the contents of the `fgACL_MarbleSampleCC.zip` file to the `fgACL_MarbleSampleCC` directory. The contents of the `fgACL_MarbleSampleCC` directory will be the `fgACL_Operations.go`, `fgGroups_Operations.go`, `fgMarbles_chaincode.go` ,`fgResource_Operations.go`, and `go.mod` files and the `oracle.com` directory.
   - From the command line, go to the `fgACL_MarbleSampleCC` directory, and run `GO111MODULE=on go mod vendor`. This command downloads the required dependencies and adds them to the `vendor` directory.
   - Compress all the contents (the four Go files, the `go.mod` file, and the `vendor` and `oracle.com` directories) of the `fgACL_MarbleSampleCC` directory in ZIP format. Your chaincode is ready to be deployed to Blockchain Platform.
3. Hyperledger Fabric v1.4.7 only: Generate the chaincode package that will be deployed to Blockchain Platform:
   - Install govendor:

     ```
     go get -u github.com/kardianos/govendor
     ```

   - Extract the contents of `fgACL_MarbleSampleCC.zip` to the `fgACL_MarbleSampleCC` directory. The contents of the `fgACL_MarbleSampleCC` directory would be: `fgACL_Operations.go`, `fgGroups_Operations.go`, `fgMarbles_chaincode.go`, `fgResource_Operations.go` and the `vendor` directory.
   - From a command line, go to the `fgACL_MarbleSampleCC` directory, and run `govendor sync`. This will download the required dependency (`github.com/op/go-logging`) and add it to the `vendor` directory.
   - Compress all the contents (the four Go files and the `vendor` directory) of the `fgACL_MarbleSampleCC` directory in ZIP format. Your chaincode is ready to be deployed to Blockchain Platform.
4. Install and deploy the updated sample chaincode package (`fgACL_MarbleSampleCC.zip`) as described in Use Quick Deployment.
5. On the **Developer Tools** tab, open the **Application Development** pane, and then follow the instructions to download the Node.js SDK.
6. On the **Developer Tools** tab, open the **Application Development** pane, and then click **Download the development package**.
   a. Extract the development package into the same folder with the Node.js files downloaded with the sample.
   b. In the `network.yaml` file, look for the `certificateAuthorities` entry and its `registrar` entry. The administrator's password is masked (converted to `***`) in the `network.yaml` when downloaded. It should be replaced with the administrator's clear text password when running this sample.
7. Register a new identity with your Blockchain Platform instance:
   a. Create a new user in IDCS (referred to as `<NewIdentity>` in the following steps) in the IDCS mapped to your tenancy.

b. Give this user the `CA_User` application role for your instance.

### Implement the Fine-Grained Access Control Marble Sample

The following steps will enroll your new user and implement the ACL restrictions using the provided Node.js scripts.

1. **Enroll the new user:**

```
node registerEnrollUser.js <NewIdentity> <Password>
```

2. **Initialization:** Initialize the access control lists.

```
node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
<ChaincodeName> ACLInitialization
```

3. **Create the access control lists, groups, and resources:** This creates the ACL resources described in the overview.

```
node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
<ChaincodeName> createFineGrainedAclSampleResources
```

4. **Create your test marble resources:** This creates several test marble assets - blue1 and blue2 owned by tom, red1 and red2 owned by jerry, and green1 and green2 owned by spike.

```
node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
<ChaincodeName> createTestMarbles
```

### Testing the Access Control

In order to test that our access control lists are only allowing the correct users to perform each function, we'll run through some sample scenarios.

1. **Transfer a marble:** We're transferring marble `blue1` from tom to jerry. Since there are no restrictions on who can transfer a single marble, this should complete successfully.

```
node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
<ChaincodeName> transferMarble blue1 jerry
```

2. **Transfer a marble as the administrative user:** We're transferring marble `blue1` from jerry to spike. Since there are no restrictions on who can transfer a single marble, this should also complete successfully.

```
node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
<ChaincodeName> transferMarble blue1 spike
```

3. **Get history:** Now we'll query the history of the marble named `blue1`. It should return that it was transferred first to jerry then to spike.

```
node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
<ChaincodeName> getHistoryForMarble blue1
```

4. **Transfer all red marbles:** The `redMarblesAcl` ACL should allow this transfer because the newly registered identity has the required `"redMarblesTransferPermission=true"` Fabric attribute, so the two red marbles should be transferred to tom.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red tom
   ```

5. **Transfer all red marbles as the administrative user:** The administrative identity doesn't have the `"redMarblesTransferPermission=true"` Fabric attribute, so the `redMarblesAcl` ACL should block this transfer.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red jerry
   ```

6. **Transfer all green marbles:** By default, only explicitly defined access is allowed. Because there isn't an ACL which allows for bulk transfer of green marbles, this should fail.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor green tom
   ```

7. **Delete a marble:** The `deleteMarbleAcl` ACL allows this deletion because the newly registered identity has the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> delete green1
   ```

8. **Delete a marble as the administrative user:** The `deleteMarbleAcl` ACL should prevent this deletion because the administrative identity doesn't have the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js < AdminIdentity > <Password> <ChannelName>
   <ChaincodeName> delete green2
   ```

### Sample Files Reference

These tables list the methods available in the chaincode and application files included with the sample.

**fgMarbles_chaincode.go**

| Function | Description |
|---|---|
| `initMarble` | Create a new marble |
| `transferMarble` | Transfer a marble from one owner to another based on name |
| `createTestMarbles` | Calls `initMarble` to create new sample marbles for testing purposes |
| `createFineGrainedAclSampleResources` | Creates the fine-grained access control list (ACL), groups, and resources required by our test scenario |

| Function | Description |
|---|---|
| transferMarblesBasedOnColor | Transfers multiple marbles of a certain color to another owner |
| delete | Delete a marble |
| readMarble | Returns all attributes of a marble based on name |
| getHistoryForMarble | Returns a history of values for a marble |

### fgACL_Operations.go

| Methods | Parameters | Description |
|---|---|---|
| getACL | • name | Get a named ACL or read all ACLs. The user invoking the method must have READ access to the named ACL. |
| createACL | • name<br>• description<br>• accesses<br>• patterns<br>• allowed<br>• BindACLs<br>• Identity_Certificate | To create a new ACL, the user invoking the method needs to have CREATE access to the bootstrap resource named "._ACLs". Duplicate named ACLs are not allowed |
| deleteACL | • name | The user invoking the method must have DELETE access to the named ACL. |
| updateACL | • name<br>• description<br>• accesses<br>• patterns<br>• allowed<br>• BindACLs | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| addAfterACL | • aclName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| addBeforeACL | • aclName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| addPatternToACL | • aclName<br>• BindPattern | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| removePatternFromACL | • aclName<br>• BindPattern | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| updateDescription | • aclName<br>• newDesc | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |

| Methods | Parameters | Description |
|---|---|---|
| removeBindACL | • aclName<br>• bindAclNameToRemove | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| addAccess | • aclName<br>• accessName | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| removeAccess | • aclName<br>• accessName | The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist. |
| ACLInitialization | • none | This function is used to initialize the fine-grained ACL support. |

### fgGroups_Operations.go

| Methods | Parameters | Description |
|---|---|---|
| getGroup | • name | If name="GetAll", it returns all the groups the identity has access to. Otherwise, it returns the individual group details (if accessible) based on name.<br>The user invoking the method must have READ access to this group. |
| createGroup | • name<br>• description<br>• patterns<br>• bindACLs | Returns success or error.<br>The user invoking the method must have CREATE access to bootstrap group ". Group" |
| deleteGroup | • name | The user invoking the method must have DELETE access to this group. |
| addAfterGroup | • groupName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to this group. |
| addBeforeGroup | • groupName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to this group. |
| updateDescriptionForGroup | • groupName<br>• newDesc | The user invoking the method must have UPDATE access to this group. |
| removeBindAclFromGroup | • groupName<br>• bindAclNameToRemove | The user invoking the method must have UPDATE access to this group. |
| addMembersToGroup | • groupName<br>• pattern | The user invoking the method must have UPDATE access to this group. |

| Methods | Parameters | Description |
|---|---|---|
| removeMembersFromGroup | • groupName<br>• pattern | The user invoking the method must have UPDATE access to this group. |

### fgResource_Operations.go

| Methods | Parameters | Description |
|---|---|---|
| createResource | • name<br>• description<br>• bindACLs | The user invoking the method needs to have CREATE access to the bootstrap resource named ". Resources". Duplicate named resources are not allowed. |
| getResource | • name | The user invoking the method must have READ access to the resource |
| deleteResource | • name | The user invoking the method must have DELETE access to the named resource |
| addAfterACLInResource | • ResourceName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to this resource |
| addBeforeACLInResource | • ResourceName<br>• existingBindAclName<br>• newBindAclName | The user invoking the method must have UPDATE access to this resource |
| updateDescriptionInResource | • ResourceName<br>• newDesc | The user invoking the method must have UPDATE access to this resource |
| removeBindACLInResource | • ResourceName<br>• bindAclNameToRemove | The user invoking the method must have UPDATE access to this resource |
| checkResourceAccess | • ResourceName<br>• access | Checks whether the current user invoking the method has the specified access to the named resource. |

C

# Run Solidity Smart Contracts with EVM on Oracle Blockchain Platform

You can run Solidity smart contracts with an Ethereum Virtual Machine (EVM) deployed as a chaincode on Oracle Blockchain Platform.

The EVM runs Solidity smart contracts in Ethereum networks. The EVM was created through the Hyperledger Burrow project and integrated into Hyperledger Fabric. This project enables you to use a Hyperledger Fabric permissioned blockchain platform to interact with Ethereum smart contracts written in an EVM-compatible language such as Solidity.

The following steps outline the process of running a Solidity smart contract on a provisioned Oracle Blockchain Platform:

1. Download the EVM chaincode package from the Oracle Blockchain Platform console.
2. Deploy the EVM chaincode on a channel.
3. Generate bytecode for a Solidity smart contract by using the Remix IDE.
4. Deploy the smart contract bytecode into the deployed EVM chaincode. Use the address returned from the deployment to send transactions.

Steps in this topic have been tested with the EVM chaincode package that is available from the Oracle Blockchain Platform console, and might not work with other releases.

> **Note:**
>
> If your chaincode was previously installed on a Hyperledger Fabric v1.4.7 instance, it should continue to work as expected when your instance is upgraded to Hyperledger Fabric v2.x.

### Download the EVM Chaincode and Fab3 Package

On the **Developer Tools** tab of the Oracle Blockchain Platform console, open the **Application Development** pane and then click **Download the EVM chaincode package.** You must be an admin user to download the file.

### Deploy EVM Chaincode on Oracle Blockchain Platform

After you download the EVM chaincode package, you deploy it on Oracle Blockchain Platform.

1. Log into the Oracle Blockchain Platform console.
2. On the **Chaincodes** tab, click **Deploy a New Chaincode**.
3. Select **Quick Deployment**, and enter the following information:
   - **Package Label:** enter a description of the chaincode package.
   - **Chaincode Language:** GoLang.

- **Chaincode Name:** enter the name of the chaincode. For example, enter `soliditycc`.
- **Version:** `v1`.
- **Init-required:** leave this unselected.
- **Channel:** select the channels where you want to install the chaincode.
- **Is Packaged Chaincode:** leave this unselected.
- **Chaincode source:** upload the `evmcc.zip` package that you downloaded previously.

For more details on the Quick Deployment wizard and restrictions on fields such as **Package Label** and **Chaincode Name**, see: Use Quick Deployment.

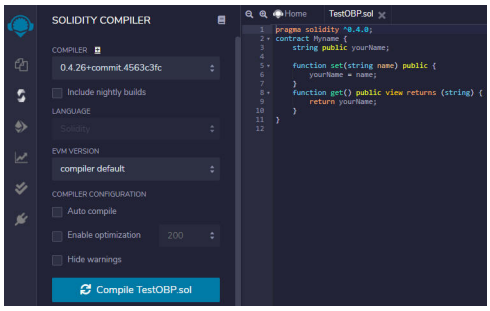
After you submit your information, the EVM chaincode is visible in the **Chaincodes** tab and is listed as a deployed chaincode on each channel that you selected to install it on.

### Create and Compile Your Solidity Smart Contract

1. Open the browser-based Remix IDE: https://remix.ethereum.org/.
2. If you already have a Solidity smart contract written, import it into Remix.
3. If you don't have a Solidity smart contract written, create a Solidity file (`.sol`) in Remix and do one of the following:
   - If you're familiar with Solidity you can create your own smart contract file.
   - You can use the Simple Storage sample code provided in the Solidity documentation: Solidity: Introduction to Smart Contracts
   - You can use the sample code being used for this example, which takes `string name` as an input and prints the same as output string using `set(name)` and `get()`:

```
pragma solidity ^0.4.0;
contract Myname {
    string public yourName;

    function set(string name) public {
        yourName = name;
    }
    function get() public view returns (string) {
        return yourName;
    }
}
```

   You might see an error message about the default compiler version not matching the version that you've specified in your smart contract.
4. Compile your smart contract. Open the Solidity Compiler panel in Remix, ensure that your smart contract tab is open to select it as the file being compiled, set the compiler version to the most recent 4.X version, and click Compile.




5. After the file is compiled, click the **Bytecode** icon to copy the bytecode as a JSON document to your clipboard.
6. Paste the copied bytecode into a text editor and save it.

### Deploy the Smart Contract

In the copied bytecode, the section you need is the `"object"` field. This is the EVM bytecode of a sample smart contract.


```
"object": "6080604052348015610010576000 80fd5b506104108061002060003960 00f30060
80604052600436106100575760 00357c0100000000000000000000000000000000000000000000000000
0000000000900463
ffffffff1680634ed3885e1461005c5780636d4ce63c146100c5578063d97d663014610155575
b600080fd5b34801561
006857600080fd5b506100c360048036038101908080359060200190820180359060200190808
0601f01602080910402
602001604051908101604052809392919081815260200183838082843782019150505050509
1929192905050506101
e5565b005b3480156100d157600080fd5b506100da6101ff565b604051808060200182810382
5283818151815260200
9150805190602001908083836000 5b838110156101a578082015181840152602081019050610
0ff565b505050509050
90810190601f1680156101475780820380516001836020036101000a031916815260200191505
b509250505060405180
910390f35b3480156101616157600080fd5b5061016a6102a1565b604051808060200182810382
5283818151815260200
9150805190602001908083836000 5b838110156101aa578082015181840152602081019050610
18f565b505050509050
90810190601f1680156101d7578082038051600183602003610100 0a031916815260200191505
b509250505060405180
910390f35b8060009080519060200190 6101fb92919061033f565b5050565b6060600080546001
8160011615610100 02
0316600290048060 1f016020809104026020016040519081016040528092919081815260200018
2805460018160011615
6101000203166002900480156102975780601f1061026c576101008083540402835291602001 9
1610297565b82019190
6000526020600020905b81548152906001019060200180831161027a57829003601f168201915
b505050505090509056
5b60008054600181600116156101000203166002900480601f016020809104026020016040519
0810160405280929190
8181526020018280546001816001161561010002031660029004801561033375780601f1061030
c576101008083540402
8352916020019161033756 5b820191906000526020600020905b81548152906001019060200 18
```

```
0831161031a57829003
601f168201915b505050505081565b828054600181600116156101000203166002900490
0600052602060002090601f01
602090048101928260 1f106103805780516 0ff191683800117855561 03ae565b8280016
00101855582156103ae579182
015b828111156103ad57825182559160200191906001019061039 2565b5b5090506103b
b91906103bf565b5090565b61
03e191905b808211156103dd576000816000905550600101610 3c5565b5090565b90560
0a165627a7a72305820a990d4
0b57c66329a32a18e847b3c18d6c911487ffadfed2098e71e8cafa0c980029",
```

In general, the EVM expects two arguments:

- The `to` address.
- The `input` that's necessary in Ethereum transactions.

To deploy smart contracts, the `to` field is the zero address, and the `input` is the compiled EVM bytecode of the contract. Thus, there are two arguments provided to the `invoke` command. The first one, which was traditionally supposed to be a function name inside the chaincode, is now `0000000000000000000000000000000000000000`, and the second argument is the Solidity smart contract bytecode.

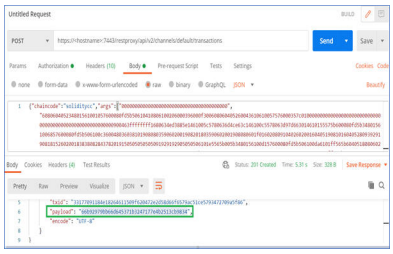
1. To deploy the Solidity smart contract on Oracle Blockchain Platform, you can make the following REST proxy call to send the two arguments to the EVM.

   ```
   {
       "chaincode": "<evmcc-ccid>",
       "args": [
           "0000000000000000000000000000000000000000",
           "<bytecode-of-the-smart-contract>"
       ],
       "timeout": 0,
       "sync": true
   }
   ```

   The following example uses cURL to deploy the Solidity smart contract to Oracle Blockchain Platform with the name `solidityc c`:

   ```
   curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
   <channelname>/transactions' \
   -H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
   -H 'Content-Type: application/json' \
   --data-raw '{"chaincode":"<evmcc-ccid>","args":
   ["0000000000000000000000000000000000000000","<bytecode-of-the-smart-
   contract>"],"timeout":0,"sync":true}'
   ```

2. The response `payload` of the transaction is the contract address for your deployed contract. Copy this address and save it. The contract address is used when you run smart contract functions.



In this example, the smart contract address is `66b92979bb66d645371b3247177e4b2513cb9834`.

There are two ways to interact with a deployed smart contract:

1. By using a hash value of the method and input parameters.
2. By using the method name and input parameters directly.

### Interacting With the Smart Contract by Using Hash Values

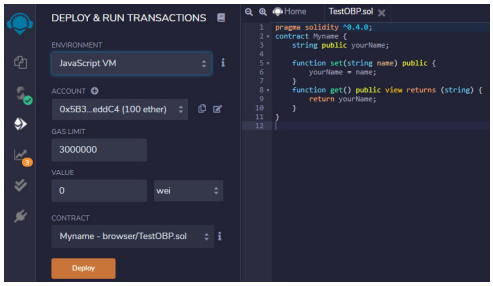
After you have the smart contract address, you can use the following calls to interact with the deployed smart contract via the REST proxy.

To execute functions, you use invoke and query transactions but with different parameters. The sample contract contains two functions: `get` and `set`.

In these transactions, the `to` field is the contract address and the `input` field is the function execution hash concatenated with any of the required arguments.

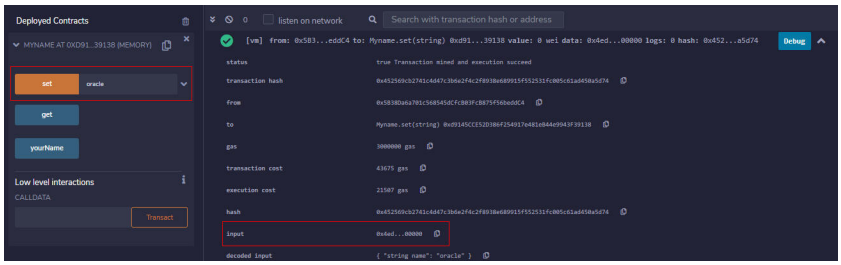
You need to acquire the hash of the function execution to run a transaction. A simple way to do this is to execute the functions in the Remix IDE and to then copy the hash from the transaction logs:

1. In the Remix IDE, open the **Deploy and Run Transactions** panel. Ensure that your contract is selected in the **Contract** field, and click **Deploy**.




After the deployment completes, the contract should be listed in the **Deployed Contracts** list.

2. Expand the contract in the **Deployed Contracts** list. The smart contract functions are listed.
3. Run a transaction. For the provided example, enter `oracle` and then click **set**.
4. The Terminal window shows the transaction logs. If the transaction logs are minimized, expand them by clicking the log. Copy the value of the `input` field (which is the function execution hash) by clicking the icon next to it. Save this value to the same location as your contract address, removing the leading `0x`.




5. After you have the function execution hash and the contract address, you can run the set transaction on Oracle Blockchain Platform using the hash and address as the raw data arguments.

```
--data-raw '{"chaincode":"<chaincodename>","args":
["<contractaddress>","<setfunctionexecutionhash>"]}'
```

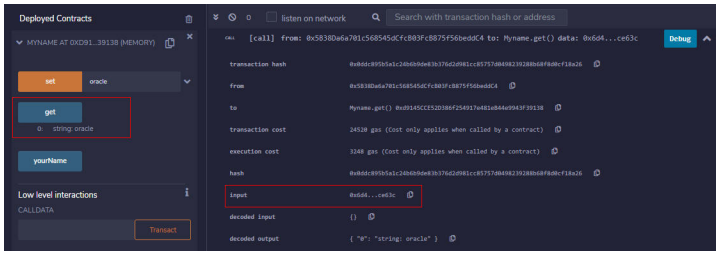
For example, using cURL:

```
curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
<channelname>/transactions' \
-H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
-H 'Content-Type: application/json' \
--data-raw '{"chaincode":"solidityсc","args":
["66b92979bb66d645371b3247177e4b2513cb9834","4ed3885e000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000066f7261636c650000000000000
0000000000000000000000000000000000000000000000000"]}'
```

6. Open the Oracle Blockchain Platform console and check that the transaction is listed in the ledger.

To run another transaction such as a query by using the smart contract's `get` function, you can generate the function execution hash in Remix and then combine it with the contract address:

1. In Remix on the **Deploy and Run Transactions** panel, ensure that your contract is still listed under **Deployed Contracts**. If not, redeploy it.
2. Click **get**. Retrieve and save the input from the transaction as you did with the **set** transaction, removing the leading `0x`.



3. You can use this transaction hash and the contract address to run a query transaction against the chaincode deployed on Oracle Blockchain Platform.

```
--data-raw '{"chaincode":"<chaincodename>","args":
["<contractaddress>","<getfunctionexecutionhash>"]}'
```

For example in cURL:

```
curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
<channelname>/chaincode-queries' \
-H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
-H 'Content-Type: application/json' \
--data-raw '{"chaincode":"solidityccc","args":
["66b92979bb66d645371b3247177e4b2513cb9834","6d4ce63c"]}'
```

The returned payload will contain the asset being queried - in the example case the string `oracle`.

The following sample payload illustrates another way to send a transaction using input encoded in hexadecimal format.

```
{
    "chaincode": "<evmcc-ccid>",
    "args": [
        "<smart-contract-address>",
        "<hexadecimal-encoded-method-and-input-parameters>"
    ],
    "sync": true
}
```

### Interacting With the Smart Contract by Using Method Names

Use the optional `setAbi` method to set an Application Binary Interface (ABI) specification, which allows you to send input and get output in readable (not hexadecimal-encoded) format instead of bytecode. Click the **ABI** icon (next to the **Bytecode** icon) on the Solidity Compiler panel in the Remix IDE to get the ABI specification. The following input types are supported for the readable input format: `uint`, `string`, `address`, `bool`, `uint[]`, `string[]`, and `address[]`. Running transactions on smart contracts that implement method overriding and method overloading is not supported from the `fab3` proxy, only from the REST proxy. The following sample payload uses the `setAbi` method to set an ABI specification.

```
{
    "chaincode": "<evmcc-ccid>",
    "args": [
        "setABI",
        "<smart-contract-address>",
        "<abi-spec-of-smart-contract>" --> use the string format of the abi
specification
    ],
    "sync": true
}
```

You can also make calls using the method name and input parameters directly, as shown in the following sample payload.

```
{
    "chaincode": "<evmcc-ccid>",
    "args": [
        "<smart-contract-address>",
        "<smart-contract-method-name>",
        "[<array-of-input-parameters>]", --> empty array if there are
no input parameters.
        "[<string-array-of-input-types>]" --> this is optional and is
needed when there is method overriding in the smart contract.
    ],
    "sync": true
}
```

### Configuring Gas Amounts

You can configure the gas amount by using the `setGasAmount` and `getGasAmount` methods, as shown in the following sample payloads.

```
{
    "chaincode": "<evmcc-ccid>",
    "args": [
        "setGasAmount",
        "<gas-amount>"
    ],
    "sync": true
}
```

```
{
    "chaincode": "<evmcc-ccid>",
    "args": [
        "getGasAmount"
    ],
    "sync": true
}
```

# Configuring the Fab3 Proxy

In the Ethereum/EVM development world, many clients and wallets use the `web3` library to deploy and manage smart contracts in Ethereum networks.

The `web3` library invokes the Ethereum JSON RPC API, which must be available through a `web3` provider.
The `fab3` proxy is a `web3` provider, which exposes a set of the Ethereum JSON RPC APIs and facilitates the use of a web3-based client with the EVM chaincode. The `fab3` proxy uses the Hyperledger Fabric Go SDK to connect and interact with the Oracle Blockchain Platform `evmcc` chaincode.

The following steps guide you through setting up the environment to use the `web3` library and the `fab3` proxy to deploy and interact with smart contracts in Ethereum Virtual Machine (EVM) chaincode.

- Before you configure the `fab3` proxy, you must follow all of the steps to set up the EVM chaincode. See Run Solidity Smart Contracts with EVM on Oracle Blockchain Platform. The EVM chaincode and `fab3` package contains the connection profile, including the `network.yaml` file and artifacts.
- The following steps apply only to Oracle Blockchain Platform instances running on Hyperledger Fabric v2.x.

1. On the **Developer Tools** tab of the service console, open the **Application Development** pane, and then click **Download Fab3 configuration including connection profile.** You must be an admin user to download the file.
2. Extract the files from the package that you downloaded.
3. Export the admin credentials from the service console.
   a. On the **Network** tab, click the **More Actions** button for your organization in the **Organizations** table.
   b. Click **Export Admin Credential**.
   c. Click **OK** to save the credentials archive file.
   d. Extract the downloaded file.
4. Copy the admin certificate (`.pem` file) that you extracted in the previous step to the following locations, substituting the actual organization and user IDs in the paths:

   ```
   ./artifacts/crypto/peerOrganizations/<organization-id>/users/<user-id>/msp/signcerts/
   ./artifacts/crypto/peerOrganizations/<organization-id>/users/<user-id>/msp/keystore/
   ```

5. Set up the environment variables that are required for the `fab3` proxy. For more information about the required environment variables, see *Setting up the Fab Proxy* at EVM Smart Contracts.

   ```
   export FAB3_CONFIG= # Path to the network.yaml in the extracted EVM chaincode and fab3 package
   export FAB3_USER= # User identity being used for the proxy (Matches the users names in the crypto-config directory specified in the config)
   export FAB3_ORG= # Organization of the specified user
   export FAB3_CHANNEL= # Channel to be used for the transactions
   export FAB3_CCID= # ID of the EVM Chaincode deployed in your fabric network
   export FAB3_PORT=5000 # Port the proxy will listen on. If not provided, the default is 5000.
   ```

6. Open a terminal window in the folder where you extracted the `fab3` package. In the Hyperledger EVM Smart Contracts documentation, follow the steps in the *Building the Fab Proxy* section to build the fab proxy and in the *Connecting to the Proxy* section to install `web3` and connect to the proxy.

You can now follow the steps in the *Deploying a Contract* and *Interacting with a Deployed Contract* sections of the Hyperledger EVM Smart Contracts documentation to deploy and interact with smart contracts using the `web3` library.

D

# Updating Applications for Hyperledger Fabric v2.x

When you upgrade the platform version, you might need to make changes to your existing applications so that they work with the new version of Hyperledger Fabric.

Multiple versions of the Hyperledger Fabric SDKs are available. Use a version of the SDK that is compatible with the version of Hyperledger Fabric that your instance is based on. For instances based on Hyperledger Fabric v2.x, use versions that are compatible with the Hyperledger Fabric v2.2 long-term support (LTS) release. Oracle Blockchain Platform was verified to work with the following versions for Hyperledger Fabric v2.x:

- Hyperledger Fabric client SDK for Node.js version 2.2.9
- Hyperledger Fabric client SDK for Java version 2.2.2
- Hyperledger Fabric client SDK for Go version 1.0.0

Hyperledger Fabric v2.x requires Go version 1.20 or later, so you might need to upgrade the version of Go that you use.

> **Note:**
>
> You might encounter timeout errors on queries if you deploy an existing chaincode `.zip` file again and the indexes are stored in the root directory of the chaincode package, instead of under the `META-INF` directory. To avoid timeout errors, ensure that indexes in existing chaincode that you deploy to a Hyperledger Fabric v2.x instance are in the following directory:
>
> ```
> META-INF/statedb/relationaldb/indexes
> ```
>
> For more information, see State Database Indexes.

You might need to update your application if it uses a client SDK to complete more complex operations such as managing chaincode life cycles, listening for events, or managing digital wallets, as discussed in the following sections.

### Hyperledger Fabric SDK for Node.js

The following table summarizes the differences between versions 1.4 and 2.x of the Hyperledger Fabric SDK for Node.js. For more information, see Migrating client applications from v1.4 to v2.0 in the Hyperledger Fabric documentation.

| Change in version 2.x | Customer action |
|---|---|
| The `fabric-client` module was removed. | Refactor applications to use the `fabric-network` module. |

| Change in version 2.x | Customer action |
|---|---|
| Wallets, used for storing and accessing identity information, were redesigned. | See the Hyperledger Fabric documentation for guidelines and utilities to migrate wallets. |
| The event listener API and behavior were redesigned. | Rewrite the event listeners to use the new API. |
| The SDK no longer provides administrative and management capabilities, include the ability to create channels and manage chaincode life cycles. | Use the command-line interface for these operations. Existing clients that use version 1.4 functions for life cycle management will not work with a Hyperledger Fabric v2.x instance. |

The following table lists the classes that are available in the Hyperledger Fabric v2.x `fabric-network` module versus the Hyperledger Fabric v1.4.7 `fabric-network` module.

| Version 2.x fabric-network module classes | Version 1.4 fabric-network module classes |
|---|---|
| `DefaultCheckpointers`<br>`Gateway`<br>`HsmX509Provider`<br>`IdentityProviderRegistry`<br>`Transaction`<br>`Wallet` | `AbstractEventHubSelectionStrategy`<br>`AbstractEventListener`<br>`BaseCheckpointer`<br>`BaseWallet`<br>`CommitEventListener`<br>`Contract`<br>`ContractEventListener`<br>`CouchDBWallet`<br>`EventHubManager`<br>`FileSystemCheckpointer`<br>`FileSystemWallet`<br>`Gateway`<br>`HSMWalletMixin`<br>`InMemoryWallet`<br>`Network`<br>`Query`<br>`RoundRobinEventHubSelectionStrategy`<br>`Transaction`<br>`X509WalletMixin` |

### Hyperledger Fabric SDK for Java

If your application uses lifecycle APIs that are now deprecated in the Java SDK for Hyperledger Fabric 2.0 (InstallProposalRequest, InstantiateProposalRequest, and UpgradeProposalRequest), rewrite your application to use the APIs in the newer version of the SDK. For more information, see Java SDK for Hyperledger Fabric 2.0 release notes.

### Oracle Blockchain Platform REST API (REST Proxy)

No changes are needed to invoke existing chaincodes.

To support the initialization function for newer chaincodes that require it, an optional `isInit` parameter was added to the existing transaction API. For more information, see Send a Transaction in the REST API documentation.

### Oracle Blockchain Platform REST API (Console)

Update the version used in all API calls. For Hyperledger Fabric v1.4.7, the API version number is 1.1. For Hyperledger Fabric v2.x, the API version number is 2.

For example, to get the list of installed chaincodes for Hyperledger Fabric v1.4.7, use the following REST endpoint:

```
/console/admin/api/v1.1/chaincodes
```

To get the list of installed chaincodes for Hyperledger Fabric v2.x, use the following REST endpoint:

```
/console/admin/api/v2/chaincodes
```

> **Note:**
>
> Although most of the existing APIs have a new Hyperledger Fabric v2.x equivalent, there is not a 1:1 match. Some APIs are unique to each version of Hyperledger Fabric, and some have different parameters for each release. For example, on Hyperledger Fabric v2.x the deployment functions have moved from the `chaincode` subpath to the `channel` subpath. For more information, see New, Changed and Deprecated APIs in the REST API documentation.

Because Hyperledger Fabric v2.x includes a new chaincode life cycle with new procedures for installing chaincode on peers and starting it on a channel, you might need to update any related API calls. To learn more, see Chaincode Life Cycle.