Oracle® Cloud

Oracle Blockchain Platform Digital Assets Edition

G16572-04
March 2025

Oracle Cloud Oracle Blockchain Platform Digital Assets Edition,

G16572-04

Contents

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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

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.
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 Overview

Oracle Blockchain Platform Digital Assets Edition is an extension of Oracle Blockchain Platform.

Oracle Blockchain Platform Digital Assets Edition

Oracle Blockchain Platform Digital Assets Edition is an extension of Oracle Blockchain Platform. Digital Assets Edition extends the transformative capabilities of blockchain into the realm of digitally native assets, enabling organizations to create, manage, and transact tokenized representations of physical and intangible assets. The platform empowers financial institutions and other businesses to build end-to-end applications quickly and cost-effectively using a scalable and secure architecture. Built on Hyperledger Fabric based permissioned blockchain infrastructure, it ensures a secure, scalable, and enterprise-ready environment for managing the complete lifecycle of digital assets - from issuance and transfer to tracking and settlement.

Oracle Blockchain Platform Digital Assets Edition extends the general-purpose blockchain platform with new features and supplies pre-built applications and components for use in scenarios such as central bank digital currency (CBDC) and digital bond marketplaces as well as numerous other digital asset use cases.

What does Oracle Blockchain Platform Digital Assets Edition include?

Oracle Blockchain Platform Digital Assets Edition includes the following components:

- An enhanced version of Oracle Blockchain Platform with added features and components to facilitate deploying comprehensive digital assets solutions. Oracle Blockchain Platform stores data on a permissioned blockchain based on the Linux Foundation open-source Hyperledger Fabric project.
- An enhanced version of Blockchain App Builder, which assists with rapid development, testing, debugging, and deployment of secure chaincodes that handle multiple digital assets. Blockchain App Builder also provides tools for automated generation of wrapper APIs from any generated chaincode, including the related Terraform scripts to deploy them along with a Postman collection for quick testing of the APIs. For information about the standard version of Blockchain App Builder, see Build Chaincodes with Low-Code Blockchain App Builder. For information about the enhancements, see Blockchain App Builder Enhancements.
- Prepackaged chaincodes for two domain-specific applications: wholesale central bank digital currency (CBDC) and a bond marketplace, along with chaincode specification templates for deposit token, generic fungible token, generic non-fungible token, and combined token applications.
- Wrapper API packages for the wholesale CBDC and bond marketplace samples. These also contain a Terraform script that provisions all of the necessary OCI resources, as well as a Postman collection to quickly test the APIs.
- A sample UI application for wholesale CBDC, based on Oracle Visual Builder. Oracle Visual Builder is a cloud-based software development Platform as a Service (PaaS). The sample application includes support for multiple personas.

- Sample analytics dashboards for wholesale CBDC, based on Oracle Analytics Cloud. The sample analytics dashboards provide real-time insights into wholesale CBDC transactions.

What Are the Advantages of Oracle Blockchain Platform Digital Assets Edition?

With its unique focus on digital asset tokenization and lifecycle management, it simplifies complex processes such as compliance, asset transfers, and ownership verification while ensuring real-time tracking and auditability.

Key features tailored to Oracle Blockchain Platform Digital Assets Edition include:

Pre-Built Application-Specific Templates
Tailored templates and modules reduce the complexity of implementing blockchain solutions across diverse industries and use cases. These features empower you by embedding domain expertise directly into the platform, enabling faster time to market and reducing dependence on external expertise.

Low-Code Development Frameworks
The platform simplifies application creation with user-friendly tools and pre-configured smart contracts. This approach ensures you can rapidly design, test, and deploy solutions, reducing time to market and enhancing agility in a competitive environment. By lowering entry barriers, you can focus on creating value and innovating without being hindered by technical overhead.

Scalable and Cost-Effective Architecture
The platform infrastructure eliminates the need for extensive initial investments by offering ready-to-use solutions that scale with organizational needs. This reduces operational costs and makes the digital assets innovation accessible to all.

Integrated Governance and Security Features
Built-in permissioned access and governance tools streamline digital assets adoption while maintaining robust security protocols. Role-based access control provides a granular permission model to secure privileges to perform token lifecycle operations and approval models aligned with best practices for financial operations.

Modular Design that Accelerates Innovation
The platform supports iterative development with a composable architecture and modular design, enabling real-time updates and seamless integration of advanced features. This flexibility drives innovation, allowing you to stay ahead of market trends and evolving customer demands.

Benefits of Oracle Blockchain Platform Digital Assets Edition

Instant domain expertise

- Instant access to specialized domain expertise enables faster market entry
- Avoid high costs of hiring experts for digital assets
- Leverage financial expertise to create digital asset solutions without the steep blockchain learning curve

Rapid application development

- Shorten development cycles especially when building applications from scratch
- Ability to leverage an agile process due to quicker development cycles
- Prepackaged components and frameworks reduce development time and maintain flexibility

Cost-effective development

- Reduce high development costs for extensive in-house development resources and overhead
- Reduce resource limitations for smaller teams

Lower barriers to entry for smaller businesses

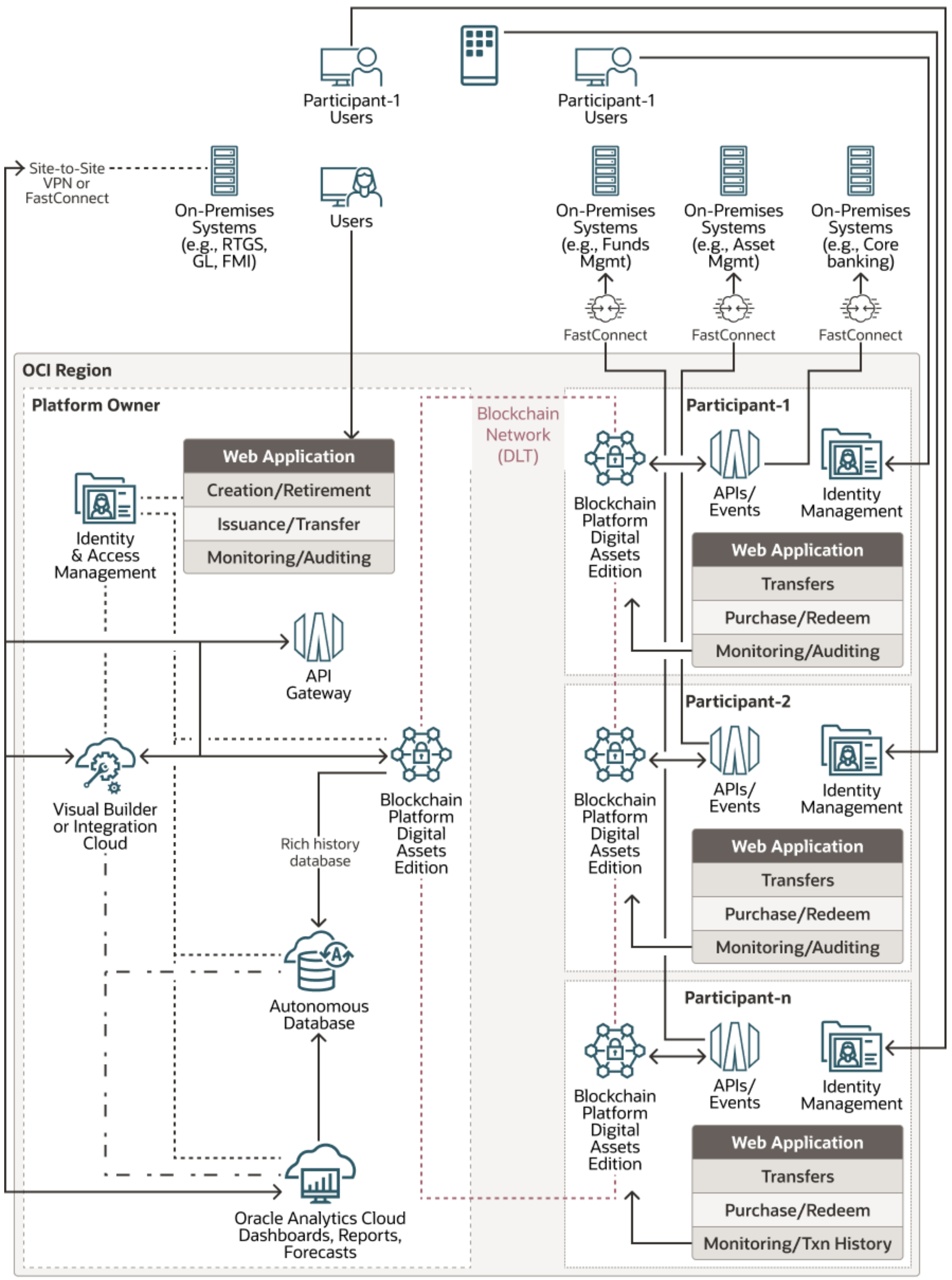
- Overcome high entry barriers due to high capital expenditure, need for specialized expertise, and lengthy product development

Accelerated innovation

- Streamlined processes speed up innovation cycles

Built-in secure controls

- Role-based access control provides protection from unauthorized access of senstive data and systems
- Protects your brand and customer loyalty



Additional Services and Applications

The following services and applications might be required to use some functions of Oracle Blockchain Platform Digital Assets Edition.

You can incur additional costs based on usage of these services.

- The wrapper APIs use the following Oracle Cloud Infrastructure (OCI) services:
  - **Oracle API Gateway service**: The OCI API Gateway service enables you to publish APIs with private endpoints that are accessible from within your network, and which you can expose with public IP addresses if you want them to accept internet traffic.
    See: Overview of API Gateway
  - **Virtual Cloud Network (VCN)**: A virtual, private network that you set up in Oracle data centers.
    See: Overview of VCNs and Subnets.
  - **Identity and Access Management (IAM) Policies**: IAM policies govern control of resources in Oracle Cloud Infrastructure (OCI) tenancies.
    See: IAM Policies Overview.
  - **OCI Functions**: When you have written the code for a function and it's ready to deploy, you can use a single Fn Project command to perform all the deploy operations in sequence.
    See: Overview of Functions
  - **OCI Registry (OCIR)**: Container Registry makes it easy for you to store, share, and manage container images (such as Docker images).
    See: Overview of Container Registry
- The rich history database uses the following application:
  - **Oracle Autonomous Database**: Oracle Blockchain Platform Digital Assets Edition used Oracle Autonomous Database to store its rich history database. Oracle Analytics then accesses this data to populate the pre-built analytics dashboard.
    See: About Autonomous Database on Dedicated Exadata Infrastructure.
- **Visual Builder**: Oracle Blockchain Platform Digital Assets Edition includes pre-built Oracle Visual Builder applications that are specific to industries and domains such as central bank digital currency (CBDC) and NFT marketplaces.
- **Oracle Analytics Cloud**: Oracle Blockchain Platform Digital Assets Edition includes prebuilt dashboards that use Oracle Analytics Cloud.
- **Oracle REST Data Services**: The analytics dashboards also require Oracle REST Data Services.

These must be provisioned on OCI in the same tenancy as your Oracle Blockchain Platform Digital Assets Edition instance.

Each additional service or application may incur additional costs while being used. For pricing estimates, you can enter your services and resources into the OCI cost estimator: OCI Service Cost Estimator

API Gateway

Oracle Blockchain Platform Digital Assets Edition uses API Gateway service to deploy wrapper APIs generated by Blockchain App Builder.

Wrapper API endpoints are published on API Gateway by using a Resource Manager stack on Oracle Cloud Infrastructure (OCI). Stacks deploy and manage groups of cloud resources in a predefined and repeatable manner, which can simplify the orchestration and automation of infrastructure provisioning. The OCI API Gateway service enables you to publish APIs with private endpoints that are accessible from within your network, and which you can expose with public IP addresses if you want them to accept internet traffic.

Oracle Blockchain Platform REST API endpoints require you to pass parameters such as the chaincode name, timeout and sync values, and arguments including the method name every time you invoke the API. In scenarios where the parameters are common to all methods, you can use wrapper APIs to invoke the API using only the method arguments. Each chaincode method will have a distinct wrapper API endpoint. In some cases, multiple REST API endpoints from different chaincodes and also Oracle Identity Cloud Service endpoints can be merged to form a single wrapper API endpoint.

The following example shows calling the `associateTokenToAccount` method with Oracle Blockchain Platform.

Endpoint: `https://<blockchain_instance>:7443/restproxy/api/v2/channels/default/transactions`

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "associateTokenToAccount",

"oaccount~78b47483e4033a0c6be3b678080264e7967d53f56d4b024edd96eb8957c452d4",
        "t2"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

With the wrapper APIs supported by Oracle Blockchain Platform Digital Assets Edition, the same invocation looks like the following example:

Endpoint: `https://<blockchain_instance>/appbuilder/associate`

```
{

"accountId":"oaccount~efc22a0316a47dd06679920140717b686c7366a627a707c5e8c89015
bb21796a",
 "tokenId":"t1"
}
```

API Gateway is configured as part of a stack deployment when you deploy your wrapper APIs as described in Deploy Wrapper APIs.

Oracle Visual Builder

Oracle Visual Builder is a cloud-based software development Platform as a Service (PaaS) and a hosted environment for your application development infrastructure. It provides an open-source standards-based solution to develop, collaborate on, and deploy applications within Oracle Cloud.

Oracle Blockchain Platform Digital Assets Edition includes a pre-built Oracle Visual Builder application for the wholesale central bank digital currency (CBDC) scenario.

For more information, see Wholesale CBDC Sample Application.

Oracle Autonomous Database

Oracle Blockchain Platform Digital Assets Edition uses Oracle Autonomous Database to store its rich history database. Oracle Analytics then accesses this data to populate the pre-built analytics dashboard.

For details on how to set up and use the rich history database, see Create the Rich History Database.

Oracle Analytics Cloud

Oracle Blockchain Platform Digital Assets Edition uses Oracle Analytics Cloud to generate pre-built dashboards and data visualizations for the wholesale central bank digital currency (CBDC) scenario.

For more information, see Wholesale CBDC Sample Analytics Package.

2

Create an Instance

As an Oracle Cloud Infrastructure administrator, you can create and set up an Oracle Blockchain Platform Digital Assets Edition instance for your organization.

Before You Create Your Instance

Before you set up Oracle Blockchain Platform Digital Assets Edition using Oracle Cloud Infrastructure Console, Oracle recommends that you take some time to plan your service.

Create a Cloud Account

Activate your Oracle Blockchain Platform Digital Assets Edition and sign in to Oracle Cloud Infrastructure.

- Sign up for your Oracle Cloud Account
- Sign In For the First Time

Create a Compartment

When you sign up for Oracle Cloud Infrastructure, Oracle creates your tenancy with a root compartment that holds all your cloud resources. You then create additional compartments within the tenancy (root compartment) and corresponding policies to control access to the resources in each compartment. Before you create an Oracle Blockchain Platform instance, Oracle recommends that you set up the compartment where you want the instance to belong.

You create compartments in Oracle Cloud Infrastructure Identity and Access Management (IAM). See:

- Setting Up Your Tenancy
- Managing Compartments

Plan Your Instance

Editions and Shapes

There are three shapes of Oracle Blockchain Platform Digital Assets Edition available:

- Medium: 8 OCPUs, 150 GB storage, 4 peers (additional can be added later)
- Large: 16 OCPUs, 150 GB storage, 6 peers (additional can be added later)
- X-Large: 32 OCPUs, 150 GB storage, 6 peers (additional can be added later)

If you create a Digital Assets shape, you can scale your Blockchain Platform up or down resulting in a new shape called Digital Assets Custom. See Scale Your Instance for details.

Platform Versions

You can also select which platform version to use, which will determine if your network is running on Hyperledger Fabric v1.4, v2.2, or v2.5. Oracle Blockchain Platform Digital Assets Edition only supports Hyperledger Fabric v2.5 or later.

Billing

Oracle Blockchain Platform OCPU-based meters use high precision billing, meaning that you are billed per second. The minimum billing amount is for one minute; any instance running for less than one minute will still be charged for one minute of time.

For pricing estimates, you can enter your services and resources into the OCI cost estimator: OCI Service Cost Estimator

Create an Oracle Blockchain Platform Digital Assets Edition Instance Using the Console

As an Oracle Cloud Infrastructure administrator, you can create and set up an Oracle Blockchain Platform Digital Assets Edition instance for your organization.

There are two types of Oracle Blockchain Platform instances you can provision:

- **Founder organization**: a complete blockchain environment, including a new network to which participants can join later on.
- **Participant instance**: if there is already a founder organization you want to join, you can create a participant instance if your credentials provide you with access to the network.

1. Sign in to your Oracle Cloud Infrastructure account. You must sign in as a federated Oracle Identity Cloud Service user.
2. In the Console, click the **Navigation** menu in the top-left corner.
3. Under **Developer Services**, select **Blockchain Platform**.
4. From the **Compartment** list, select the compartment in which you want to create the service.
5. Click **Create Blockchain Platform**.
6. Enter a name for your Oracle Blockchain Platform instance.
   The service instance name:
   - Must contain one or more characters.
   - Must not exceed 15 characters.
   - Must start with an ASCII letter: `a` to `z` or `A` to `Z`.
   - Must contain only ASCII letters or numbers.
   - Must not contain a hyphen.
   - Must not contain any other special characters.
   - Must be unique within the identity domain.
7. Optionally enter a description of your instance.
8. Select your platform version. This specifies the version of Hyperledger Fabric you want your instance to run on. The founder and participants must all be on the same version of Hyperledger Fabric.
9. Select if you are creating a new network, or creating a participant instance to join an existing network.
10. Select the Oracle Blockchain Platform Digital Assets edition and the appropriate shape. The editions and shapes are described in Plan Your Instance.
11. Oracle Blockchain Platform includes a certificate authority (CA), which is used to create self-signed certificates for all blockchain nodes in your instance.

If you want to use certificates from your own certificate authority and use the Oracle Blockchain Platform certificate authority as an intermediary CA, you can upload your CA archive. The certificate you upload will be used to sign the intermediary certificates for Oracle Blockchain Platform nodes, thus including them under your root CA chain.

The archive is a zip file which contains the following files:

- CA chain - named `ca-chain.pem`. The entire CA file sequence from the signing CA to the top-level CA should be present.
- key - named `ca-key.pem`. The key should be a 256-bit elliptic curve key. The `prime256v1` curve is recommended.
- certificate - named `ca-cert.pem`

The archive must be less than 2MB. The files must directly reside inside the zip archive such that when the archive (.zip) is unzipped, the files are visible in the current directory at the same level as the archive (.zip) file. The files should not be present inside a nested directory inside the archive.

12. If you want to use tags for your instance, expand the Advanced Options section and add your tags.
13. Click **Create**.

It takes about 15 minutes to create the service. Display the **Instance** page to check the current status.

Verify Your Instance

Navigate to your service in the Oracle Cloud Infrastructure console, and sign in to verify that your Oracle Blockchain Platform Digital Assets Edition instance is up and running.

For more information about signing into the Oracle Cloud Infrastructure console, see Signing In to the Console.

1. On the Oracle Cloud Infrastructure console, click the **Navigation** menu in the top-left corner.
2. Under **Developer Services**, select **Blockchain Platform**.
3. From the **Compartment** list, select the compartment that you used to create the instance.
4. Click the name of the new instance.
5. Click **Service Console**. This launches the Oracle Blockchain Platform console.

Now that your instance is created, you can manage the instance itself as described in Manage the Lifecycle of an Instance.

You can also manage users, access and permissions as described in Set Up Users and Application Roles.

Digital Assets Page

All of the Oracle Blockchain Platform Digital Assets Edition functionality is available from the Digital Assets page in the console.

After you create an instance, click the **Digital Assets** tab to open the Digital Assets page. You can access the following panes from the Digital Assets page.

Wholesale CBDC Application
You can use the Wholesale CBDC Application pane to complete the following tasks. For more information, see Wholesale Central Bank Digital Currency Application.

- Install, deploy, and invoke the wholesale CBDC sample chaincode online.
- Download the wholesale CBDC sample chaincode package.
- Download the wholesale CBDC wrapper API package.
- Download database view definitions, which you can use for analytics and the pre-built web application.
- Download the sample web application, built in Oracle Visual Builder, and workbooks for use with Oracle Analytics Cloud.

Bond Marketplace Application
You can use the Bond Marketplace Application pane to complete the following tasks. For more information, see Bond Marketplace Application.

- Download the bond marketplace sample chaincode package.
- Download the bond marketplace wrapper API package.

Fungible Token Framework
You can use the Fungible Token Framework pane to complete the following tasks.

- Download the fungible token framework sample chaincode package. For more information, see Fungible Token Framework Chaincode Package.
- Download the fungible token framework wrapper API package. For more information, see Fungible Token Framework Wrapper API Package.

Non-Fungible Token Framework
You can use the Non-Fungible Token Framework pane to complete the following tasks.

- Download the non-fungible token framework sample chaincode package. For more information, see Non-Fungible Token Framework Chaincode Package.
- Download the non-fungible token framework wrapper API package. For more information, see Non-Fungible Token Framework Wrapper API Package.

Combined Token Framework
You can use the Combined Token Framework pane to complete the following tasks.

- Download the combined token framework sample chaincode package. For more information, see Combined Token Framework Chaincode Package.
- Download the combined token framework wrapper API package. For more information, see Combined Token Framework Wrapper API Package.

3

Wholesale Central Bank Digital Currency Application

The wholesale central bank digital currency (CBDC) sample application can support transaction scenarios between a central bank and other financial institutions.

The wholesale CBDC sample represents the life cycle of a wholesale CBDC token. The sample specification file uses the extended Token Taxonomy Framework standard that is supported by Blockchain App Builder. The generated chaincode includes methods from initializing a wholesale CBDC token, account operations, and role assignments to minting, transfers, and burning of CBDC tokens. It also provides notary accounts for approving minting, transfers, and burning operations, and it supports compliance through account-level daily limits and auditing procedures.

The wholesale CBDC solution comprises the following downloadable packages.

- Blockchain App Builder specification file
- Pre-built wholesale CBDC chaincode
- Wrapper APIs for CBDC setup and lifecycle operations
- Oracle Database view definitions for columnar views of the JSON transaction data in rich history database tables and use by analytics workbooks
- A sample application using Oracle Visual Builder, and a sample Oracle Analytics dashboard

To get the wholesale CBDC packages, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Wholesale CBDC**.

Wholesale CBDC Chaincode Package

Oracle Blockchain Platform Digital Assets Edition includes sample chaincode for the wholesale central bank digital currency (CBDC) scenario.

The wholesale CBDC chaincode package is based on the extended Token Taxonomy Framework standard supported by Blockchain App Builder. The package implements methods for a system where tokens representing fiat currency are held at financial institutions and are issued and managed by regulated financial institutions. The sample specification file generates methods for initializing tokens, managing accounts, assigning roles, and completing operations such as minting, transferring, and burning tokens. It also provides notary accounts for approving minting and transfers, enforces compliance with account-level daily limits, and enables auditing procedures. The chaincode can be generated by Blockchain App Builder from the `WholesaleCBDC.yml` specification file in either TypeScript or Go.

The wholesale CBDC chaincode package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `WholesaleCBDC.zip`, an archive file that contains the packaged chaincode for deployment.
- `WholesaleCBDC.yaml`, a specification file that you can use with Blockchain App Builder to scaffold the `WholesaleCBDC` chaincode.

- `WholesaleCBDC_postman_collection.json`, a Postman collection that enables you to test the APIs in the chaincode.
- `README.md`, a step-by-step guide for working with the chaincode.

To get the wholesale CBDC chaincode package, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Wholesale CBDC Application**.

You can try out the wholesale CBDC chaincode in the same way as other samples. You must be an administrator to install and deploy sample chaincodes.

1. On the Wholesale Central Bank Digital Currency page, click **Install**.
2. In the Install Chaincode window, specify one or more peers to install the chaincode on, and then click **Install**.
3. Click **Deploy**.
4. In the Deploy Chaincode window, specify the channel to deploy the chaincode to. The initial parameters of the chaincode will be used to specify the token administrator. These values are not editable. They default to the organization ID and user ID of the user who is logged in to the console. Click **Deploy**.
5. Click **Invoke**.
6. In the Invoke Chaincode window, specify the channel to run the transaction on. In the **Action** list, specify an action to complete. Click **Execute**.

Specification File

The wholesale CBDC specification file (`Wholesale_CBDC.yml`) is based on the extended Token Taxonomy Framework specification file. It includes a `model` attribute, which generates the application-specific chaincode. In this case, `model: wcbdc` creates additional methods for the wholesale CBDC application when the chaincode is generated. The following snippet from the specification file shows the `model` attribute.

```
#
# Copyright (c) 2024, Oracle and/or its affiliates. All rights reserved.
#

assets:

# Token asset to manage the complete lifecycle of Wholesale CBDC token.

    - name: CBDC # Asset name
      type: token # Asset type
      standard: ttf+   # Token standard
      events: true # Supports event code generation for non-GET methods
      model: wcbdc # Supports creation of additional methods for Wholesale
CBDC application

      anatomy:
          type: fungible # Token type
          unit: fractional # Token unit

      behavior: # Token behaviors
          - divisible:
                decimal: 2
          - mintable:
                mint_approval_required: true
          - transferable
          - burnable:
                burn_approval_required: true
          - holdable
          - roles:
                minter_role_name: minter
                burner_role_name: burner
                notary_role_name: notary
                mint_approver_role_name: notary
                burn_approver_role_name: notary
```

Deploy and Test Wholesale CBDC Chaincode

Deploying the Chaincode

You can deploy the chaincode directly from the Oracle Blockchain Platform console or by using Blockchain App Builder. Before you deploy the chaincode, create enrollment IDs for each token user and then map the token users to their respective enrollment IDs. Specify only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to a REST Proxy in *Using Oracle Blockchain Platform*.

When you deploy the token chaincode, you must call the `init` method and pass the organization ID and user ID of the `Token Admin` user.

For information about deploying from the Oracle Blockchain Platform console, see Use Advanced Deployment in *Using Oracle Blockchain Platform*.

To deploy using Blockchain App Builder, complete the following steps.

1. Extract the `WholesaleCBDC.zip` archive file.
2. Import the `WholesaleCBDC` chaincode to the Blockchain App Builder extension in Visual Studio Code.
3. Edit the `.ochain.json` file to update the value of the `configFileLocation` key to the path of the `WholesaleCBDC.yml` specification file.
4. Open a terminal window and navigate to the chaincode folder, and then run the following command.

   ```
   npm install
   ```

For more information about deploying using Blockchain App Builder, see Deploy Your Chaincode Using Visual Studio Code in *Using Oracle Blockchain Platform*.

Sample Process Flow for the Wholesale CBDC Chaincode

A typical process flow using the wholesale CBDC methods follows these basic steps.

1. Admins use the `initializeCBDCToken` method to initialize the wholesale CBDC system.
2. Admins use the `createAccount` and `associateTokenToAccount` methods to create accounts and associate the token to accounts for all creators, central bank approvers, issuers, financial institution officers, financial institution approvers, and financial institution users in the system.
3. Admins use the `addRole` method to assign the minter role to the creator and the notary role to the central bank approver.

4. The token creator uses the `requestMint` method to submit a request to mint currency.
5. The central bank approver uses the `approveMint` method to review and approve the request to mint currency. The currency is credited to the creator's account.
6. The issuer uses the `getAccountBalance` method to verify that the credited amount is accurate.
7. The creator uses the `transferTokens` method to send currency to the issuer. The currency is credited to the issuer's account.
8. The `holdTokens` method is used to request transfer of tokens to the financial institution officer.
9. The central bank approver uses the `executeHoldTokens` method to validate and approve the transfer request. The currency is transferred to the financial institution officer's account.
10. The financial institution officer uses the `getAccountBalance` method to verify their account balance.
11. The financial institution officer uses the `holdTokens` method to request transfer of tokens to the financial institution user.
12. The financial institution approver uses the `executeHoldTokens` method to validate and approve the transfer request. The currency is transferred to the financial institution user's account.
13. The financial institution user uses the `getAccountBalance` method to verify their account balance.

For more details about using Postman collections, see the following topics in *Using Oracle Blockchain Platform*.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Wholesale CBDC Wrapper API Package

Oracle Blockchain Platform Digital Assets Edition includes a wrapper API package that extends the REST API to support operations specific to wholesale CBDC.

The wrapper API package uses the API Gateway service and OCI Functions, which are created using a Resource Manager stack on Oracle Cloud Infrastructure (OCI), to deploy API routes specifically designed for the wholesale CBDC application. The wholesale CBDC wrapper API package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `WholesaleCBDCWrapperAPI.zip`, an archive file that contains the wrapper API package including the Terraform scripts required for deployment. You deploy this file to a Resource Manager stack on OCI to create the necessary Oracle resources for the Wrapper APIs.
- `WholesaleCBDC_WrapperAPI.postman_collection.json`, a Postman collection that enables you to test the deployed wrapper APIs. The collection includes pre-configured requests with endpoints and payloads that correspond to the APIs defined in the wrapper API package.

Wrapper APIs

`activateCBDCAccount`
Original method name: `activateAccount`

This POST method activates a token account. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization. Deleted accounts cannot be activated.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"db0738d4a44f6d9c80b24fce7c518c07023f7be19edaa69b272eaf7886b4b925",
        "payload": {
            "assetType": "oaccountStatus",
            "status_id": 
"oaccountStatus~d5814d96d8517ac31727d60aace0519c58a425892ab0d378fcfb0a35771f65
ae",
            "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
            "status": "active"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 194
    }
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

`addCBAdmin`
Original method name: `addTokenAdmin`

This POST 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. The first invocation is from the admin who instantiates the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "User (Org_Id: CB, User_Id: cb) is already Token Admin."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

addCBAuditor
Original method name: `addTokenAuditor`

This POST method adds token auditors to the token chaincode. This method can be called only by the `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"cd81f6c4c9e7c18ece357dbf5c139ef66ef2d6566be3b14de5f6d0a3fd4bb2f0",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 196
    }
}
```

addFIAdmin
Original method name: `addOrgAdmin`

This method adds organization admins to the token chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"96a84dffcb9156f7271dfb414e8c43b540595044cf9145f5fd56e9873797fc4a",
        "payload": {
            "msg": "Successfully added Org Admin (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 197
    }
}
```

`addFIAuditor`
Original method name: `addOrgAuditor`

This method adds organization auditors to the token chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"44bbad35a1478cb714e32f7cfd551897868a203520aab9cea5771d3aadc1cf03",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 198
    }
}
```

`addRole`
Original method name: `addRole`

This method adds the role to the specified user and token. Account IDs are formed by creating a SHA-256 hash of the concatenated token ID, organization ID, and user ID. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "role": "role value (for example minter / burner / notary)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `role: string` – The name of the role to add to the specified 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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
```

```
"29ea766dee8e6d273eba3c40a9fea75a1aa85dc3c280d40695f6224c5c52d93c",
        "payload": {
            "msg": "Successfully added role 'notary' to Account Id:
oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8
(Org-Id: CB, User-Id: manager_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 201
    }
}
```

`approveCBDCCreation`
Original method name: `approveMint`

Notaries can call this POST method to approve a mint request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – The unique operation ID of the mint request to approve.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0",
        "payload": {
            "msg": "Successfully minted 10 tokens to Account Id:
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB, User-Id: creator_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 204
    }
}
```

`approveCBDCRetirement`
Original method name: `approveBurn`

Notaries can call this POST method to approve a burn request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – The unique operation ID of the mint request to approve.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"8252371040e41753fd843bd086692f4e1ee7d89ffa3a50bfa121c5f1565f922f",
        "payload": {
            "msg": "Successfully burned 1 tokens from account id: 
oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c 
(Org-Id: CB, User-Id: retirer_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 209
    }
}
```

`approveHoldCBDCTokens`
Original method name: `executeHoldTokens`

Notaries call this method to approve a hold on tokens, which triggers the transfer of the tokens from the payer to the payee in this business scenario. The quantity of tokens put on hold previously by the token owner is now transferred to the recipient. If the `quantity` value is less than the actual hold value, the remaining amount is available again to the owner of the token. If the `roles` behavior is specified in the `behaviors` section of the token model and the `notary_role_name` value is set, the caller account must have notary role. Otherwise, any caller with an account can function as a notary.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "quantity": 1,
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – The unique operation ID of the mint request to approve.
- `quantity: number` – The number of held tokens to transfer.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"c1149aaa486abc4931d9024c18dfcb230bb321723d1160b0bf981c0011c4856a",
        "payload": {
            "msg": "Account Id: 
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594 
(Org-Id: CB, User-Id: issuer_user_cb) is successfully executed '10' tokens 
from Operation Id '8e3145'."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 213
    }
}
```

`associateTokenToAccount`
Original method name: `associateTokenToAccount`

This POST method associates a specified account ID to a specified token. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "accountId": "account_id value",
    "tokenId": "{{bc-token-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `accountId: string` – The ID of the account.
- `tokenId: string` – The ID of the token.

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"efc7381fb6fc6174a40e83ff5f09d2bbf7f6f490365e3bbf19d5502c2cfec474",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id": 
"oaccount~1c6aa60e220b8fc70caf4cea1ed723ddb193a00321e5e0004def062816b77090",
            "user_id": "cb12",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-11T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 216
    }
}
```

`createAccount`
Original method name: `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 (`userId`) of the instance owner or the user who is logged in to the instance, the membership service provider ID (`orgId`) 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.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenType": "fungible",
    "applicationGroups": "[\"application_groups value\"]",
    "dailyLimits": 
"{\"max_daily_amount\":10000,\"max_daily_transactions\":100}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `applicationGroups: string[]` – A list of application groups the user Id belongs to, which define the user's associations in the CBDC application.
- `dailyLimits: JSON object` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id": 
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
```

```
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "application_groups": [
                    "CBDC_ADMINS"
                ],
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 100,
                "daily_transactions": 0,
                "current_date": "2024-12-09T00:00:00.000Z"
            },
            "encode": "JSON",
            "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
            "blockNumber": 188
        }
}
```

`getAllActiveCBDCAccounts`
Original method name: `getAllActiveAccounts`

This GET method returns all of the active accounts that are associated with the specified token ID.

Query:

```
/getAllActiveCBDCAccounts?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                "non_account_role_name": [
                    "token_admin"
                ],
                "role_name": null,
                "valueJson": {
                    "bapAccountVersion": 0,
                    "assetType": "oaccount",
                    "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                    "user_id": "admin_user_cb",
                    "org_id": "CB",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 0,
                    "onhold_balance": 0,
                    "application_groups": [
                        "CBDC_ADMINS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getAllSuspendedCBDCAccounts`
Original method name: `getAllSuspendedAccounts`

This GET method returns all of the suspended accounts that are associated with the specified token ID.

Query:

```
/getAllSuspendedCBDCAccounts?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                "non_account_role_name": null,
                "role_name": null,
                "valueJson": {
                    "assetType": "oaccount",
                    "bapAccountVersion": 1,
                    "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                    "user_id": "user1_fi1",
                    "org_id": "FI1",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 5,
                    "onhold_balance": 0,
                    "application_groups": [
                        "FI_CBDC_USERS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id": 
```

```
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getApproverActionHistory`
Original method name: `getActionHistory`

This GET method retrieves the history of approvals or rejections made by the caller for mint, burn, and transfer (issuance) operations, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary). This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor` or the notary.

Query:

```
/getApproverActionHistory?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "from_account_id": 
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "from_org_id": "CB",
              "from_user_id": "retirer_user_cb",
              "holding_id": "ohold~cbdc~USD~eaf6",
              "holding_status": "REJECT_BURN",
              "last_updated_time": "2024-11-26T21:43:22.000Z",
              "notary_account_id": null,
              "notary_org_id": null,
              "notary_user_id": null,
              "operation_id": null,
              "quantity": 3,
              "timetoexpiration": null,
              "to_account_id": "",
              "to_org_id": null,
              "to_user_id": null,
              "token_id": "USD",
              "token_name": null
          },
          {
              "from_account_id": 
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "from_org_id": "CB",
              "from_user_id": "retirer_user_cb",
              "holding_id": "ohold~cbdc~USD~0031",
              "holding_status": "REJECT_BURN",
              "last_updated_time": "2024-11-26T21:43:15.000Z",
              "notary_account_id": null,
              "notary_org_id": null,
              "notary_user_id": null,
              "operation_id": null,
              "quantity": 2,
              "timetoexpiration": null,
              "to_account_id": "",
              "to_org_id": null,
              "to_user_id": null,
              "token_id": "USD",
              "token_name": null
          }
      ],
      "encode": "JSON"
  }
}
```

`getCBDCAccount`
Original method name: `getAccount`

This GET method returns account details for a specified user and token. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

Query:

```
/getCBDCAccount?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-user-
id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "status": "active",
            "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON"
    }
}
```

`getCBDCAccountBalance`
Original method name: `getAccountBalance`

This GET method returns the current balance for a specified account and token. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

```
/getCBDCAccountBalance?tokenId={{bc-token-id}}&orgId={{bc-org-
id}}&userId={{bc-user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Current Balance is: 100",
            "user_balance": 100
        },
        "encode": "JSON"
    }
}
```

`getCBDCAccountsByUser`
Original method name: `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` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

```
/getCBDCAccountsByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "application_groups": [
                    "CBDC_ADMINS"
                ],
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 1000,
                "daily_transactions": 0,
                "current_date": "2024-11-20T00:00:00.000Z"
            },
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "USD",
                "token_name": "cbdc",
                "balance": 0,
                "onhold_balance": 0,
                "application_groups": [
                    "CBDC_ADMINS"
                ],
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 1000,
                "daily_transactions": 0,
                "current_date": "2024-11-20T00:00:00.000Z"
            },
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "application_groups": [
                    "CBDC_ADMINS"
                ],
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 100,
                "daily_transactions": 0,
                "current_date": "2024-12-09T00:00:00.000Z"
            }
        ],
        "encode": "JSON"
    }
}
```

`getCBDCAccountTransactionHistory`
Original method name: `getAccountTransactionHistory`

This method returns an array of account transaction history details for a specified user and token. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

```
/getCBDCAccountTransactionHistory?tokenId={{bc-token-id}}&orgId={{bc-org-
id}}&userId={{bc-user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
        ],
        "encode": "JSON"
    }
}
```

getAccountTransactionHistoryWithFilters
Original method name: `getAccountTransactionHistoryWithFiltersFromRichHistDB`

This method returns the account transaction history details from the rich history database. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

```
/getCBDCAccountTransactionHistoryWithFilters?tokenId={{bc-token-
id}}&orgId={{bc-org-id}}&userId={{bc-user-id}}&customEndpoint=custom_endpoint
value&bearerToken=bearer_token
value&filters={"pageSize":20,"bookmark":"","startTime":"2022-01-16T15:16:36+00
:00","endTime":"2022-01-17T15:16:36+00:00"}
```

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.
- `customEndpoint` – The RESTful service endpoint of the rich history database to fetch the transaction history from.
- `bearerToken` – The token to use to call the RESTful endpoint to ensure that the request is authorized.
- `filters: JSON object` – 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. The `startTime` and `endTime` properties must be specified in RFC-3339 format.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
        ],
        "encode": "JSON"
    }
}
```

getCBDCRetiredQuantity
Original method name: `getBurnQuantity`

This GET method returns the total quantity of burned tokens for a specified organization. This method can be called only by a `Token Admin`, `Token Auditor`, or a user with the burner role.

```
/getCBDCRetiredQuantity?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": {
          "burnt_quantity": 31
      },
      "encode": "JSON"
  }
}
```

getNetCBDCTokens
Original method name: `getNetTokens`

This GET 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. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, or an `Org Admin` or `Org Auditor`.

```
/getNetCBDCTokens?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Net supply of token for Token Id: USD is 878 tokens.",
            "quantity": 878
        },
        "encode": "JSON"
    }
}
```

getOnHoldIds
Original method name: `getOnHoldIds`

This GET method returns a list of all of the holding IDs for a specified account. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `AccountOwner` of the account.

```
/getOnHoldIds?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-user-
id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Holding Ids are:
ohold~cbdc~USD~8e3147,ohold~cbdc~USD~8e315",
            "holding_ids": [
                "ohold~cbdc~USD~8e3147",
                "ohold~cbdc~USD~8e315"
            ]
        },
        "encode": "JSON"
    }
}
```

getPendingCBDCIssuance
Original method name: `getPendingIssuance`

This method retrieves all pending issuance (transfer) transactions where the caller is assigned as an approver, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary). This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor`, or the `Notary`.

```
/getPendingCBDCIssuance?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "from_org_id": "CB",
              "from_user_id": "creator_user_cb",
              "holding_id": "ohold~cbdc~USD~8e314",
              "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e314",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id":
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          },
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id":
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
              "from_org_id": "CB",
              "from_user_id": "issuer_user_cb",
              "holding_id": "ohold~cbdc~USD~8e315",
              "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e315",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id": 
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          }
      ],
      "encode": "JSON"
  }
}
```

getPendingCBDCRequest
Original method name: `getPendingRequest`

This method retrieves all pending requests of a specified type where the caller is assigned as an approver. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode or the `Notary`.

```
/getPendingCBDCRequest?tokenId={{bc-token-id}}&requestType=request_type value
```

Parameters:

- `tokenId: string` – The ID of the token.
- `requestType: string` – The transaction type. For example, `mint` or `burn`.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "valueJson": {
                  "assetType": "ohold",
                  "holding_id": "ohold~cbdc~USD~op123",
                  "operation_id": "op123",
                  "token_id": "USD",
                  "token_name": "cbdc",
                  "operation_type": "mint",
                  "status": "pending",
                  "from_account_id": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_id": "",
                  "notary_account_id": 
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
                  "quantity": 10,
                  "time_to_expiration": "0",
                  "category": "category value",
                  "description": "description value"
              }
```

```
            }
        ],
        "encode": "JSON"
    }
}
```

`getTotalBalanceByCallerOrgId`
Original method name: `getTotalBalanceByCallerOrgId`

This method retrieves the total balance of the caller's organization. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or any account owner.

```
/getTotalCBDCBalanceByCallerOrgId
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "totalBalance": 704
          }
      ],
      "encode": "JSON"
  }
}
```

`getTotalCreatedCBDCTokens`
Original method name: `getTotalMintedTokens`

This method returns the total number of minted tokens for a specified token. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Query:

```
/getTotalCreatedCBDCTokens?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Total minted token for Token Id: USD is 910 tokens.",
            "quantity": 910
        },
        "encode": "JSON"
    }
}
```

getTransactionWithBlockNumber
Original method name: `getTransactionWithBlockNumber`

This GET method returns the details of the transaction for the specified transaction ID.

Query:

```
/getTransactionWithBlockNumber?tokenId={{bc-token-
id}}&transactionId=transaction_id value
```

Parameters:

- `tokenId: string` – The ID of the token.
- `transactionId: string` – The ID of the transaction.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "blockNo": 82,
              "key":
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
              "metadata": null,
              "txnNo": 0,
              "value": null,
              "valueJson": {
                  "assetType": "otransaction",
                  "transaction_id":
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
                  "token_id": "USD",
                  "from_account_id": "",
                  "from_account_balance": 0,
                  "from_account_onhold_balance": 0,
                  "to_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_balance": 100,
                  "to_account_onhold_balance": 0,
                  "transaction_type": "REQUEST_MINT",
                  "amount": 200,
                  "timestamp": "2024-11-20T06:48:42.000Z",
                  "number_of_sub_transactions": 0,
                  "holding_id": "",
                  "sub_transaction": "false",
                  "description": ""
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getUserByCBDCAccountId`
Original method name: `getUserByAccountId`

This method returns user details (`orgId`, `userId`, and `tokenId`) for a specified account. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, or an `Org Admin` or `Org Auditor` of the specified organization.

Query:

```
/getUserByCBDCAccountId?accountId=account_id value
```

Parameters:

- `accountId: string` – The ID of the account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "token_id": "USD",
            "user_id": "admin_user_cb",
            "org_id": "CB"
        },
        "encode": "JSON"
    }
}
```

`getUsersByRole`
Original method name: `getUsersByRole`

This GET method returns a list of all users for a specified role and token. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode.

Query:

```
/getUsersByRole?tokenId={{bc-token-id}}&role=role value (for example minter /
burner / notary)
```

Parameters:

- `tokenId: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "users": [
                {
                    "token_id": "USD",
                    "user_id": "creator_user_cb",
                    "org_id": "CB"
                },
                {
                    "token_id": "USD",
                    "user_id": "cb4",
                    "org_id": "CB"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

`holdCBDCTokens`
Original method name: `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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "notaryOrgId": "notary_org_id value",
    "notaryUserId": "notary_user_id value",
    "quantity": 1,
    "timeToExpiration": "time_to_expiration value",
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `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.
- `notaryOrgId: string` – The membership service provider (MSP) ID of the notary in the current organization.
- `notaryUserId: string` – The user name or email ID of the notary.
- `quantity: number` – The number of tokens to put on hold.
- `timeToExpiration` – 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`.
- `infoDetails: JSON` – The description and category as shown in the following example.

  ```
  {
     "category" : "category input",
     "description" : "description input"
  }
  ```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"e575d339299bb98afe83207e749cd07654f209673c84c6973738b6094da33062",
        "payload": {
            "msg": "AccountId
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594
(Org-Id: CB , User-Id: issuer_user_cb) is successfully holding 10 tokens"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 211
    }
}
```

`init`
Original method name: `init`

This method is called when the chaincode is deployed. The user information is saved as the `Token Admin` of the chaincode.

Payload:

```
{
    "adminList": "[{\"org_id\":\"{{bc-org-id}}\",\"user_id\":\"{{bc-admin-
user}}\"}]"
}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"fdb7dc89832c8045a333823b77fa24ae628178148dc93b3550040e070d7cd807",
        "payload": "",
        "encode": "UTF-8",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 263
    }
}
```

`initializeCBDCToken`
Original method name: `initializeCBDCToken`

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.

Payload:

```
{
    "tokenAsset": "{\"token_id\":\"{{bc-token-id}}
\",\"token_desc\":\"token_desc value\",\"Currency_Name\":\"Currency_Name
value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <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.

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a JSON representation of the token asset that was created.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"aa7a4f4cc214e1a041a5a6fb7ca7530f08256559e538c9f9582e6fd12c9e65c8",
        "payload": {
            "assetType": "otoken",
            "events": false,
            "token_id": "t1",
            "token_name": "cbdc",
            "token_desc": "token_desc value",
            "token_standard": "ttf+",
            "token_type": "fungible",
            "token_unit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "holdable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner",
                "notary_role_name": "notary",
                "mint_approver_role_name": "notary",
                "burn_approver_role_name": "notary"
            },
            "mintable": {
                "max_mint_quantity": 1000,
                "mint_approval_required": true
            },
            "burnable": {
                "burn_approval_required": true
            },
            "divisible": {
                "decimal": 2
            },
            "currency_name": "currency_name value",
            "token_to_currency_ratio": 999
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 267
    }
}
```

rejectCBDCCreation
Original method name: `rejectMint`

This method can be called by a minter notary to reject a minting request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to reject minting.
- `operationId: string` – The unique operation ID that represents the mint request.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22",
        "payload": {
            "msg": "Successfully rejected mint request with Operation Id
'op1234' to mint 10 tokens of token id USD"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 205
    }
}
```

rejectCBDCRetirement
Original method name: `rejectBurn`

This method can be called by a notary to reject a burning request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to reject for burning.
- `operationId: string` – The unique operation ID that represents the burn request.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"474a08183986c84fe321aa925670539db3b1bc90b02fa65956ad8c771fff5bbe",
        "payload": {
            "msg": "Successfully rejected burn request with Operation Id
'burn1234' to burn 10 tokens of token id USD"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 210
    }
}
```

`rejectHoldCBDCTokens`
Original method name: `releaseHoldTokens`

This POST 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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"f04ba8895d52bc636d843f88476002bc99d01480c36be87c8fa259cd47a29380",
        "payload": {
            "msg": "Successfully released '10' tokens from Operation Id 
'8e3144' to Account Id: 
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594 
(Org-Id: CB, User-Id: issuer_user_cb)."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 214
    }
}
```

`removeCBAdmin`
Original method name: `removeTokenAdmin`

This POST method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. An admin cannot remove themselves.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"6a3b9b568d04b5beb29830f91efe4e8c6310b6cf36940cecfb4ab690fbfde739",
        "payload": {
            "msg": "Successfully removed Token Admin (Org_Id: CB, User_Id:
```

```
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 218
    }
}
```

`removeCBAuditor`
Original method name: `removeTokenAuditor`

This POST method removes a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"a886a6040fbc76374a3c78c89ab0ffc9f7b8391cc5239b169bf3b878cf40c67b",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id: 
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 219
    }
}
```

`removeFIAdmin`
Original method name: `removeOrgAdmin`

This POST method removes a user as a `Org Admin` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e2a634f6093f89b1984e20ff86a513fabb7c3ade7cc9e27d9734b4aaf6c88597",
        "payload": {
            "msg": "Successfully removed Org Admin (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 220
    }
}
```

`removeFIAuditor`
Original method name: `removeOrgAuditor`

This POST method removes a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"c3bc720461004a53b37c68d4bb264858b88d980bc093a0a3ebb62a32974fb306",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id: 
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 221
    }
}
```

`removeRole`
Original method name: `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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "role": "role value (for example minter / burner / notary)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: 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.
- `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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a message with account details.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"274f0d0a2c4c3929817fb85b2e857519695c3c238ccf9903b084b87e9be7ee12",
        "payload": {
            "msg": "Successfully removed role 'notary' from Account Id:
oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8
(Org-Id: CB, User-Id: manager_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 200
    }
}
```

`requestCBDCCreation`
Original method name: `requestMint`

This method can be called by a minter to send a request to the minter notary to create a specified amount of tokens.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "notaryOrgId": "notary_org_id value",
    "notaryUserId": "notary_user_id value",
    "quantity": 1,
    "timeToExpiration": "time_to_expiration value",
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to mint.
- `operationId: string` – The unique operation ID that represents the mint request.
- `notaryOrgId: string` – The membership service provider (MSP) ID of the minter notary who will process the request.
- `notaryUserId: string` – The user name or email ID of the minter notary who will process the request.
- `quantity: number` – The amount of tokens to mint.

- `timeToExpiration` – The time after which the minting request expires and is no longer valid.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

```
{
     "category" : "category input",
     "description" : "description input"
}
```

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e",
        "payload": {
            "msg": "AccountId
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB , User-Id: creator_user_cb) has successfully submitted request to
mint 10 tokens"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 202
    }
}
```

`requestCBDCRetirement`
Original method name: `requestBurn`

This method can be called by a burner to send a request to the notary to destroy a specified amount of tokens.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "notaryOrgId": "notary_org_id value",
    "notaryUserId": "notary_user_id value",
    "quantity": 1,
    "timeToExpiration": "time_to_expiration value",
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to burn.
- `operationId: string` – The unique operation ID that represents the burn request.
- `notaryOrgId: string` – The membership service provider (MSP) ID of the burner notary who will process the request.
- `notaryUserId: string` – The user name or email ID of the burner notary who will process the request.
- `quantity: number` – The amount of tokens to burn.
- `timeToExpiration` – The time after which the burning request expires and is no longer valid.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"0adb57ca9776c8760468c40465e0f0d37472f0b4b32fd02561ef28b3f7b28cf1",
        "payload": {
            "msg": "AccountId
oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c
(Org-Id: CB , User-Id: retirer_user_cb) has successfully submitted request to
burn 10 tokens"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 206
    }
}
```

`setApplicationGroups`
Original method name: `setApplicationGroups`

This POST method is used to set the `application_groups` parameter in the account details for the specified application groups. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenId": "{{bc-token-id}}",
    "applicationGroups": "[\"application_groups value\"]",
    "endorsers": {{endorsers}}
}
```

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.
- `applicationGroups: string[]` – A list of application groups the user ID belongs to, defining the user's associations in the CBDC application.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a success message with the quantity of tokens burned and the account ID.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "",
            "token_name": "",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-09T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 188
    }
}
```

`setMaxDailyAmount`
Original method name: `setMaxDailyAmount`

This POST method is used to set the `maxDailyAmount` parameter in the account details for the specified amount. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenId": "{{bc-token-id}}",
    "maxDailyAmount": 1,
    "endorsers": {{endorsers}}
}
```

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.
- `maxDailyAmount: number` – The maximum daily amount value for the specified account, which defines the maximum amount that can be transacted daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"28682e0564e4721b6c1a8ec106f8c5c98319e9439959dbb9f83d8e6f111d9975",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 222
    }
}
```

`setMaxDailyTransactionCount`
Original method name: `setMaxDailyTransactionCount`

This POST method is used to set the `maxDailyTransactions` parameter in the account details for the specified amount. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenId": "{{bc-token-id}}",
    "maxDailyTransactions": 1,
    "endorsers": {{endorsers}}
}
```

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.
- `maxDailyTransactions: number` – The maximum daily amount value for the specified account, which defines the maximum number of transactions allowed per day.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"8b6fb01de697562ee098110054f05d4a314933bd11ef471991cb43e25b68bad9",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 223
    }
}
```

`suspendCBDCAccount`
Original method name: `suspendAccount`

This method suspends a fungible token account. It throws an error if an `accountStatus` value is not found in ledger. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
            "assetType": "oaccountStatus",
            "status_id": 
"oaccountStatus~d5814d96d8517ac31727d60aace0519c58a425892ab0d378fcfb0a35771f65
ae",
            "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
            "status": "suspended"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 195
    }
}
```

`transferCBDCTokens`
Original method name: `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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "quantity": 1,
    "infoDetails": "{\"category\":\"category 
value\",\"description\":\"description value\"}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.

- `toOrgId: string` – The membership service provider (MSP) ID of the receiver (payee) in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `quantity: number` – The number of tokens to transfer.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

```
{
     "category" : "category input",
     "description" : "description input"
}
```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"d613b2494b965811b2fa2106152b7085f2d6d7d43e949b10b8668722d3636fe7",
        "payload": {
            "msg": "Successfully transferred 10 tokens from account id:
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB, User-Id: creator_user_cb) to account id:
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594
(Org-Id: CB, User-Id: issuer_user_cb). Only 999 number of transactions and
1990 amount transfer left for today: 12/11/2024"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 224
    }
}
```

The wrapper API package also includes the `createCBDCAccount` API, which combines chaincode APIs and Oracle Blockchain Platform console APIs into a single endpoint for account creation.

`createCBDCAccount`
Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "applicationGroups": "[\"applicationGroups value\"]",
    "tokenId": "{{bc-token-id}}",
    "role": "role value (for example minter / burner / notary / tokenAdmin /
tokenAuditor / orgAdmin / orgAuditor)",
    "dailyLimits": "{\"max_daily_amount\":1000,
\"max_daily_transactions\":100}"
}
```

Payload parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `applicationGroups` – The application groups the account will belong to. For example, `CBDC_ADMINS`.
- `tokenId` – The ID of the token associated with the account. The ID must begin with an alphanumeric character and can include letters, numbers, and the special characters underscore (_) and hyphen (-). It cannot exceed 16 characters in length.
- `role` – The role must be one of the following values: `minter`, `burner`, `notary`, `tokenAdmin`, `tokenAuditor`, `orgAdmin` or `orgAuditor`.
- `dailyLimits` – Two numeric fields that define the daily transaction limits: `max_daily_amount` and `max_daily_transactions`.

Return Value Example:

```
"payload": {
 "msg": "Account created successfully and 'minter' role added to Account Id:
oaccount~4a86f2843c2b0105c97a77202bd9aba519c81dcef27eccc1d89125ae32770700
(Org-Id: CB, User-Id: creator_user_cb)"
 "accountDetails":
      {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~ce5c56d2068ce31b82136c8eea110a80b9251595d361db70924c4e989032a1be",
            "user_id": "creator_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_CREATORS"
            ],
            "max_daily_amount": 1000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-10-06T00:00:00.000Z"
```

```
        }
}
```

The `createCBDCAccount` API completes the following operations in sequence.

1. Creates an enrollment ID for the user in Oracle Blockchain Platform. The enrollment ID that is created is the same as the user ID, with some limitations. The ID can contain only alphanumeric characters, hyphens (-), and underscores (_). If the user ID contains any other special characters, they are replaced with an underscores (_). For example, if the user ID is adam.fripp@example.com, the enrollment ID that is created is adam_fripp_example_com. If the enrollment already exists for the specified user ID, another enrollment is not created.
2. Creates an account in the ledger with the details provided in the payload.
3. Associates the new account with the token specified in the payload.
4. Assigns the role specified in the payload to the user.

To ensure consistency and to avoid incomplete data, if any of the subsequent steps fail, the enrollment ID that was created in the first step is deleted.

Customize Wrapper APIs for Wholesale CBDC

The wholesale CBDC wrapper API is a modified version of the wrapper API package that is generated by Blockchain App Builder.

Complete the following steps if you regenerate the wrapper API package after adding custom methods to the wholesale CBDC chaincode. To ensure that your changes are compatible, you must modify the newly generated wrapper API package by using the wholesale CBDC wrapper API package that is bundled with the product.

1. Use Blockchain App Builder to generate a wrapper API package for the wholesale CBDC chaincode.
2. Extract the files from the package.
3. Copy the `createCBDCAccount` folder from the wrapper API package that is bundled with the product into the directory structure of the newly generated wrapper API package.
4. Add an entry at the end of the `terraform.vars` file for the `createCBDCAccount` method, as shown in the following example JSON string.

   ```
   \"createCBDCAccount\": {\"path\":\"/createCBDCAccount\",\"type\":
   [\"POST\"]}
   ```

   The following text shows the general format of the JSON string in the *function_path* variable in the `terraform.vars` file.

   ```
   {"<methodName>":{"path":"/<methodFolderName>","type":["<HTTP Method POST
   or GET>"]}}
   ```

5. Replace the `main.tf` file in the newly generated wrapper API package with the `main.tf` file from the wrapper API package that is bundled with the product.

Deploy and Test Wrapper APIs for Wholesale CBDC

Deploying the Wrapper API Package

Before you can deploy the wrapper API package, you must update the required configuration variables. Some configuration variables have default values, but you must manually update any variable that contains a placeholder as its default value. Configuration variables are stored in the `terraform.tfvars` file in the wrapper API archive. For more information about deploying wrapper APIs and about configuration variables, see Wrapper APIs. The following table lists the configuration variables and their default values for the wholesale CBDC wrapper API package. If the default value contains placeholders, it indicates that the user must manually provide the necessary values.

Variable name	Default value	Description
`compartment_ocid`	`<compartment_ocid>`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	`<compartment_name>`	The name of the OCI compartment.
`identity_domain`	`<identity_domain>`	The identity domain to use.
`blockchain_channel`	`<blockchain_channel>`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	`<blockchain_url>`	The Oracle Blockchain Platform URL associated with the chaincode deployment.
`blockchain_chaincode`	`WholesaleCBDC`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	`true`	The sync value to include in the payload for API calls.
`blockchain_timeout`	`6000`	The timeout value to include in the payload for API calls.
`vcn_display_name`	`WholesaleCBDC`	The display name of the OCI virtual cloud network.
`application_display_name`	`WholesaleCBDC`	The display name of the OCI application.
`gateway_display_name`	`WholesaleCBDC`	The display name of API Gateway.
`deployment_display_name`	`WholesaleCBDC`	The display name of the deployment in API Gateway.
`deployment_path_prefix`	`/WholesaleCBDC`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	`wholesalecbdc`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.

Variable name	Default value	Description
`policy_name`	`WholesaleCBDC`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

For information about the Postman collection, see Wrapper API Package Components.

Wholesale CBDC Sample Process Flow

A typical process flow using the wholesale CBDC wrapper APIs follows these basic steps.

1. The administrator uses the `initializeCBDCToken` API to initialize the wholesale CBDC system.
2. The administrator uses the `createAccount` and `associateTokenToAccount` APIs to create accounts and associate the token to accounts for all creators, central bank approvers, issuers, financial institution officers, financial institution approvers, and financial institution users in the system.
3. The administrator uses the `addRole` API to assign the minter role to the creator and the notary role to the central bank approver.
4. The token creator uses the `requestCBDCCreation` API to submit a request to mint currency.
5. The central bank approver uses the `approveCBDCCreation` API review and approve the request to mint currency. The currency is credited to the creator's account.
6. The issuer uses the `getCBDCAccountBalance` API to verify that the credited amount is accurate.
7. The creator uses the `transferCBDCTokens` API to transfer the currency to the issuer.
8. The `holdCBDCTokens` API is used to start the transfer of currency to a financial institution officer.
9. The central bank approver uses the `approveHoldCBDCTokens` API to approve the transfer of currency to the financial institution officer. The currency is credited to the financial institution officer's account.
10. The financial institution officer uses the `getCBDCAccountBalance` API to verify that the credited amount is accurate.
11. The financial institution officer uses the `holdCBDCTokens` API to start the transfer of currency to a financial institution user.
12. The financial institution approver uses the `approveHoldCBDCTokens` API to approve the transfer of currency to the financial institution user. The currency is credited to the financial institution user's account.
13. The financial institution user uses the `getCBDCAccountBalance` API to verify that the credited amount is accurate.

Postman Collection

The Postman collection in the wholesale CBDC wrapper API package includes additional attributes and APIs that support the wholesale CBDC chaincode. For more information, see Wrapper API Package Components.

Oracle Database View Definitions for Wholesale CBDC

You can use the rich history database to retrieve account transaction history and resolve incorrect balances in multiple transactions that occur in the same block.

You can use the `GetAccountTransactionHistoryWithFiltersFromRichHistDB` API to fetch account transaction history from the rich history database. When you pass the `custom_endpoint` and `bearer_token` parameters to the method, the account transaction history is retrieved from the rich history database or the state database.

To retrieve transaction history from the rich history database, you must be running Oracle Autonomous Database with Oracle REST Data Services (ORDS) and OAuth enabled.

1. Enable and configure the rich history database.

   For more information, see Enable and Configure the Rich History Database in *Using Oracle Blockchain Platform*.
2. Enable rich history on the channels that contain the chaincode data that you want to write to the rich history database. For more information, see Configure the Channels that Write Data to the Rich History Database in *Using Oracle Blockchain Platform*.
3. Download and install Node.js version 18 or later.
4. On the Digital Assets page in Oracle Blockchain Platform, select **Wholesale CBDC Application**.
5. Click **Download the Database View Definitions package**.
6. Extract the `WholesaleCBDCViewsPackage.zip` file.
7. Navigate to the `ORDSscript` folder and install the required dependencies by running the following command.

   ```
   npm install
   ```
8. Edit the `.env` file that is supplied with the script to configure it for your environment.

   Oracle REST Data Services endpoints use the following general format.

   ```
   <base_URL>/<user_name>/<resource_link>
   ```

Environment / Configuration Type	Environment / Configuration Variables	Description	Example
`DB Connection`	`CONNECTION_STRING`	The connection string for the database.	`CONNECTION_STRING="(description=(retry_count=20)(retry_delay=3)(address=(protocol=tcps)(port=1522)(host=adg.ap-sydney-1.example.com))(connect_data=(service_name=g536390e55ee33f4_db_high.adg.example.com))(security=(ssl_server_dn_match=yes)))"`
`View Configuration`	`VIEW_NAME`	The name of the view for displaying account transaction details. This can be any value that does not conflict with existing assets in the database.	`VIEW_NAME="viewTest"`
`View Configuration`	`CHAINCODE_NAME`	The name of the chaincode to fetch transaction details from in the rich history database.	`CHAINCODE_NAME="BondMarketplace"`
`View Configuration`	`INSTANCE_NAME`	The name of the instance where the chaincode is deployed.	`INSTANCE_NAME="BondMarketplace"`
`View Configuration`	`CHANNEL_NAME`	The name of the channel where the chaincode is deployed.	`CHANNEL_NAME="default"`
`ORDS Endpoint Setup`	`MODULE_NAME`	The name of the ORDS module to use. This can be any value that does not conflict with existing assets in the database.	`MODULE_NAME="demotest"`
`ORDS Endpoint Setup`	`BASE_PATH`	The base path of the ORDS URL. This can be any value that does not conflict with existing assets in the database.	`BASE_PATH="demotest"`

Environment / Configuration Type	Environment / Configuration Variables	Description	Example
`ORDS Endpoint Setup`	`PATTERN`	The pattern name of the ORDS URL. This can be any value that does not conflict with existing assets in the database.	`PATTERN="accountTransactionDetails"`
`ORDS Endpoint Setup`	`ITEMS_PER_PAGE`	The number of items to display per page in the ORDS query results. This can be any value that does not conflict with existing assets in the database.	`ITEMS_PER_PAGE=120`
`ORDS REST Endpoint`	`ORDS_REST_BASE_URL`	The base URL of the ORDS REST endpoint of the database.	`ORDS_REST_BASE_URL="https://g536390e55ee33f4_db_high.adg.ap-sydney-1.example.com"`
`Alias Configuration`	`ALIAS_NAME`	The alias to use in place of a user name in the REST endpoint URL. This can be any value that does not conflict with existing assets in the database.	`ALIAS_NAME="demotestAlias"`
`ORDS Role`	`ROLE_NAME`	The ORDS role that is assigned to the user. This can be any value that does not conflict with existing assets in the database.	`ROLE_NAME="demotest_role"`
`ORDS Privilege`	`PRIVILEGE_NAME`	The ORDS privilege that is assigned to the user. This can be any value that does not conflict with existing assets in the database.	`PRIVILEGE_NAME="demotest_priv"`
`ORDS Privilege`	`LABEL`	A label for the ORDS privilege. This can be any value that does not conflict with existing assets in the database.	`LABEL="demotest_label"`
`ORDS Privilege`	`DESCRIPTION`	A description of the ORDS privilege. This can be any value that does not conflict with existing assets in the database.	`DESCRIPTION="demotest_description"`

Environment / Configuration Type	Environment / Configuration Variables	Description	Example
`OAuth Configuration`	`CLIENT_NAME`	The client name to use for OAuth authentication with the ORDS REST endpoint. This can be any value that does not conflict with existing assets in the database.	`CLIENT_NAME="demotest_client"`
`OAuth Configuration`	`OWNER`	The owner name to use for OAuth authentication with the ORDS REST endpoint. This can be any value that does not conflict with existing assets in the database.	`OWNER="demotest"`
`OAuth Configuration`	`DESCRIPTION`	A description of the OAuth configuration. This can be any value that does not conflict with existing assets in the database.	`DESCRIPTION="demotest_description"`
`OAuth Configuration`	`SUPPORT_EMAIL`	The support email address for the OAuth configuration. This can be any value that does not conflict with existing assets in the database.	`SUPPORT_EMAIL="test@example.com"`

Analytics view names are included in the software code. The accounts view is `ACCOUNTS_MOD`. The transaction view is `TRANSACTION_MOD`. The accounts transaction view is `ACCOUNTS_TRANSACTION_MOD`.

9. Run the ORDS script by using the following command.

```
npm run start --username='<username>' --password='<password>'
```

In this example, `<username>` and `<password>` are the credentials for the rich history database, which must have the necessary permissions to create the view and the ORDS endpoint. The user name and password must be enclosed in single quotes (') because of limitations in the `bash` interface.

When the command runs, the following prompts are displayed.

`Do you want to create the View and ORDS Endpoint? (y/n)`
Enter `y` to create the view and endpoint. Enter `n` if you have already created the view and endpoint.

`Please select the language of your chaincode? (TS/GO)`
If you entered `y` previously, enter `TS` for TypeScript or `GO` for Go.

`Do you want to generate ORDS Endpoint URL and Bearer Token? (y/n)`
Enter `y` to generate the endpoint credentials. Otherwise, enter `n`.

You might see the following error.

```
Error: ORA-20049: Cannot alter the URL mapping while the schema is enabled.
Try disabling the schema first.
```

This error occurs because the database schema is enabled and thus cannot be mapped to the different alias that is specified in the `.env` file.
To work around this behavior, complete the following steps.

1. Use the same alias name that was used previously, or check the REST services in the database for the schema alias.
2. Disable the database schema and run the script again. For more information, see Oracle REST Data Services (ORDS) : Using SQL Developer.

Wholesale CBDC Sample Application and Analytics Package

Oracle Blockchain Platform Digital Assets Edition includes a sample application and Oracle Analytics workbooks for the wholesale central bank digital currency (CBDC) scenario.

Wholesale CBDC Sample Analytics Package

Oracle Blockchain Platform Digital Assets Edition includes an Oracle Analytics workbook with pre-built dashboards and data visualizations for the wholesale central bank digital currency (CBDC) scenario.

The workbook, built with Oracle Analytics, lets you visualize the currency creation, retirement, and transaction flows in the CBDC environment. After you provision the Oracle Analytics Cloud service and connect it to the Oracle Database with the custom view definitions, you can import the workbook. The workbook includes a complete set of data visualizations for the central bank and a subset with only the data relevant to the accounts and transactions at each participating institution. You can modify the data visualizations using developer mode in Oracle Analytics Cloud.

Configure Oracle Analytics Cloud

Complete the following steps to configure Oracle Analytics Cloud and use the dashboards and visualizations for the wholesale CBDC application.

1. Create an instance of Oracle Autonomous Database. For information on how to provision Oracle Autonomous Database for the wholesale CBDC sample, see Provision Autonomous Database.
2. Set up the database view definitions. For information on how to set up the database view definitions for the wholesale CBDC sample, see Oracle Database View Definitions for Wholesale CBDC.
3. Create an instance of Oracle Analytics Cloud in OCI.
   a. Log on to the Oracle Cloud Infrastructure (OCI) console.
   b. Click **Analytics & AI** > **Oracle Analytics Cloud**.
   c. Click **Create Instance** and then add the required information.
   d. Click **Create** and then wait for the instance creation to complete.

   For more information about Oracle Analytics Cloud, see Oracle Analytics Cloud.

4. Under Sample wCBDC Web Application and Analytics Workbook on the Digital Assets page in Oracle Blockchain Platform Digital Assets Edition, click **Download sample VBCS UI and sample Analytics package**. The `WholesaleCBDCAnalyticsPackage.zip` file, which contains the sample application, is downloaded to your computer.
5. Extract the `WholesaleCBDCAnalyticsPackage.zip` file, which contains the `WholesaleCBDCAnalyticsPackage` directory. The workbook (`.dva`) files are in the `WholesaleCBDCAnalyticsPackage` directory
6. Back up both workbook files, `CBDC FI Dashboard.dva` and `CBDC Central Bank Dashboard.dva`.
7. Navigate to Oracle Analytics Cloud in OCI console, and then click **Analytics Home Page**.
8. Click **Import Workbook/Flow** and import both workbook files. Enter `OraAnalytic@2025` as the password. For more information, see Import a Workbook File.
9. Update the data source connection details for the `WCBDC_Connection` dataset.
   a. Go to the Data page in Oracle Analytics Cloud.
   b. Find the `WCBDC_Connection` dataset and then click **Inspect/Edit** > **Update database connection**.
   c. Map the database views.
   d. Click **Save**.

   For more information about connecting Oracle Autonomous Transaction Processing to Oracle Analytics Cloud, see Connect to Oracle Autonomous Transaction Processing.
10. Set up roles and permissions for the central bank and financial institution dashboards.
    a. In Oracle Analytics Cloud, click **Console**.
    b. Click **Roles and Permissions**, then click **Application Roles**.
    c. Click **Create Application Role** and create the `CBRole` application role.
    d. Under Members, click **Groups** and then click **Add Groups**.
    e. Assign groups to the `CBRole` and the `BI Dataload Author` roles. The `BI Dataload Author` role is required for embedding Oracle Analytics Cloud data into Oracle Visual Builder. The Oracle Identity Cloud Service (IDCS) groups that are associated with the sample application must be mapped to the `BI Dataload Author` role. If the `CBRole` and the `BI Dataload Author` roles are not mapped, the analytics data will not be shown in the sample application, even if Oracle Analytics Cloud is integrated with Oracle Visual Builder. Map all relevant IDCS groups from the sample application to these roles. For example, map the following groups.
       - `CB_CBDC_ADMINS`
       - `CB_CBDC_AUDITORS`
       - `CB_CBDC_CREATORS`
       - `CB_CBDC_MANAGERS`
    f. Create roles for the financial institutions: `FI1Role`, `FI2Role`, and so on.
    g. Assign groups to the financial institution roles and to the `BI Dataload Author` role. You can create groups with any names, such as the following example groups. In the following example groups, change the number of the financial institution in the group name to match the corresponding financial institution role.
       - `FI1_CBDC_ADMINS`

- `FI1_CBDC_AUDITORS`
- `FI1_CBDC_MANAGERS`
- `FI1_CBDC_OFFICERS`

For more information, see Assign Application Roles to Groups.

11. Update the instance name parameters for the central bank and financial institution dashboards.
    a. Log in as an adminstrator and open the central bank dashboard.
    b. Click **Edit**.
    c. Under Parameters, right-click to edit the parameter for the central bank instance name.
    d. Enter the central bank instance name or the membership service provider (MSP) ID. For example, enter `CB`.
    e. Repeat the previous steps to update the central bank instance name in the financial institution dashboard.
12. Update the roles in the financial institution dashboard.
    a. Open the `CBDC FI` dashboard.
    b. On **Analytic View Dashboard**, click **Edit**.
    c. Click the **Data** tab.
    d. Double-click the **CB_ACCOUNT_TRANS_MOD-FI** view.
    e. Add roles for the participant and create a filter for each role. The roles that you add must also be present in the Roles and Permissions section of the dashboard. To add roles to filters, complete the following steps.
        i. On the dashboard, click **Join Diagram**.
        ii. Click the plus sign (+) to add a role.
        iii. Create a filter, and then add an **Expression Filter** that includes a query. For example, add the following SQL query, which mentions the `FI1` and `FI2` participant organizations.

        ```
        FROM_ORG_ID = 'FI1' AND TO_ORG_ID = 'FI2'
        ```
    f. Click **Console** > **Roles and Permissions**.
    g. Under Application Roles, add roles, assign user groups to roles, and verify existing roles as needed.
    h. Save your changes.
13. Move the dashboards to shared folders in Oracle Analytics Cloud.
    a. Navigate to **Catalog** > **My Folders**.
    b. Move the dashboards from `My Folders` to `Shared Folders`.
14. Test thoroughly to ensure that all visualizations, data connections, user access, and other functions are working as expected.
15. To embed the analytics dashboards in the sample Oracle Visual Builder application, complete the following steps.
    a. Register the application as a safe domain in Oracle Analytics Cloud. See Register Safe Domains.

b. Embed the analytics URL in the iFrame in the application. See Use iFrame to Embed Analytics Content into an Application or Web Page.

View Dashboard Data

After you configure Oracle Analytics Cloud for the wholesale CBDC application, you can log in and view analytics data.

All of the central bank personas that are supported by the wholesale CBDC sample application can view data from the wholesale CBDC workbook in the analytics interface. For financial institution personas, you must have the `FI_ADMINS` role or the `FI_CBDC_MANAGERS` role to access analytics data. Financial institution personas see their organization's information only.

To view the data from Oracle Analytics Cloud, complete the following steps.

1. Log in to the wholesale CBDC UI.
2. On the main page, click **Navigator Menu**.
3. In the list, select **Dashboard**. A page that displays the integrated Oracle Analytics Cloud workbooks opens.

Wholesale CBDC Sample Application

A sample wholesale central bank digital currency (CBDC) application package using Oracle Visual Builder Cloud Service is included in Oracle Blockchain Platform Digital Assets Edition, allowing you to create your own Visual Builder instance, import the provided package, and configure it.

Once you've imported the sample application, you'll update the backend configurations with your instance URLs, set up authorization, and configure Business Objects as needed.

The web application, built using Oracle Visual Builder, lets you administer and work with tokens in the wholesale CBDC life cycle. After you provision Oracle Visual Builder in your tenancy, you can import the application. The application supports role-based access by users that are defined in identity domains. Roles include admins, officers, managers, auditors, and regular users at both the central bank and at participating institutions. You can modify the application using developer mode in Oracle Visual Builder.

This structured, role-based approach ensures that every stage of the CBDC lifecycle—from minting to burning—is secure, compliant, and transparent, fostering trust and efficiency in the wholesale CBDC ecosystem.

Overview of the Sample Wholesale Central Bank Digital Currency Application

The end-to-end journey of a wholesale Central Bank Digital Currency (CBDC) application involves multiple organizational roles, each responsible for specific functions that ensure CBDC issuance, management, and transfer of CBDC tokens. This journey is facilitated by a combination of Central Bank (CB) system roles and Financial Institution (FI) organization roles, with strict approval workflows to maintain security and regulatory compliance.

The process begins with the CBDC admin who initializes the token system and creates and manages the wholesale CBDC accounts. CBDC Creator initiates the minting of CBDC tokens. Once a minting request is submitted, it moves to the CBDC Approver, who reviews and either approves or rejects the request. Upon approval, the CBDC Creator transfers the newly minted tokens to the CBDC Issuer, who is responsible for distributing these tokens to various Financial Institution (FI) Officers. This ensures that only authorized entities within the financial ecosystem receive CBDC tokens for further transactions.

At the financial institution level, the FI Admin manages and administers their specific organization, while FI Auditors have read-only access to organizational data for oversight and compliance purposes. FI Officers play a key role in managing CBDC tokens within their institutions. They receive tokens from the CBDC Issuer and can distribute them internally to FI Users or externally to officers in other organizations. However, any inter-organization or intra-organization transfers must first be approved by the FI Approver/FI Manager, who reviews and authorizes hold requests to ensure secure and compliant transactions.

In parallel, the system includes mechanisms for managing the lifecycle of CBDC tokens beyond their initial issuance. When tokens need to be removed from circulation, the CBDC Retirer initiates a burning request. Similar to the minting process, this request goes to the CBDC Approver for approval. Once approved, the CBDC burning process is completed, ensuring the reduction of CBDC supply is properly documented and regulated.

Throughout this journey, CBDC Auditors and FI Auditors maintain oversight, with read-only access to ensure transparency and compliance with regulatory standards. While the CBDC Auditor can view all the transactions involving central bank and any financial institutions, FI auditors can view the transactions where that specific FI was involved in the transaction.

Additionally, CBDC Admins oversee and manage the entire CBDC system, ensuring that all roles, transactions, and processes align with central bank policies. They are responsible for system governance, user access management, and overall operational efficiency.

Wholesale CBDC Sample Application Prerequisites

Before importing the Oracle Visual Builder sample application package, it is essential to complete several prerequisites, including the creation of all required Oracle Cloud Infrastructure (OCI) resources and Oracle Identity Cloud Service (IDCS) groups as outlined below.

Visual Builder Cloud Service

The Wholesale CBDC application sample is built using Oracle Visual Builder Cloud Service. The package needs to be imported into Visual Builder to use it.

For more information on Visual Builder, see Visual Builder.

1. Sign in to your Oracle Cloud Infrastructure account.

   Ensure you're in the correct compartment where you'll deploy the sample application.
2. In the Console, click the **Navigation** menu in the top-left corner.
3. Under **Developer Services**, select **Visual Builder**.
4. In the Visual Builder interface, click **Create Instance**.
   a. Enter an instance name and choose the default network access or another option as needed.
   b. Click **Create Visual Builder Instance**.

Once Visual Builder is provisioned, you can explore Visual Builder Designer which is the interface you'll use to interact with the wholesale CBDC sample app. See Tour the Designer.

Provision Autonomous Database

All account transaction data is stored in and fetched from the rich history database. To use the rich history database, you must create an Oracle Autonomous Database.

For additional information on the rich history database in Oracle Blockchain Platform, see Create the Rich History Database.

1. Sign in to your Oracle Cloud Infrastructure account.

   Ensure you're in the correct compartment where you'll deploy the sample application.
2. In the Console, click the **Navigation** menu in the top-left corner. Select **Oracle Database**.
3. Select Autonomous Data Warehouse, Autonomous JSON Database, or Autonomous Transaction Processing based on your workload.

   See About Autonomous Database Workload Types.
4. Click **Create Autonomous Database**.
   - **Display Name**: A user-friendly description (not unique).
   - **Database Name**: Must consist of letters and numbers only (maximum 30 characters).
   - **Workload Type:** Select **Transaction Processing**.
   - **Deployments Type:** Select the default `Serverless`.
   - **Configure the database:** Adjust the CPU and storage settings according to your requirements. However, the CBDC Application is designed to function effectively with the default values, so there is no need to modify these settings for the CBDC Application.
   - **Backup retention:** Keep the default settings.
   - **Network:**
     – **Access Type**: Select `Secure access from allowed IPs and VCNs only`.

     – **Access control list**: Select `CIDR block` from the IP notation type, and enter the value `0.0.0.0/0`.

     – **Mutual TLS (mTLS) authentication**: mTLS is not required.
5. After you select the settings, click **Create Autonomous Database**.

   When the provisioning is complete, the **Lifecycle State** will be `Available`.

Provision Oracle Blockchain Platform Digital Assets Edition

You must have an Oracle Blockchain Platform Digital Assets Edition instance provisioned for the sample application to use.

Users can create Oracle Blockchain Platform Digital Assets Edition instances with any name; however, the application supports one Central Bank as the founder of the Oracle Blockchain Platform network and six Financial Institutions (FI1, FI2, FI3, FI4, FI5, and FI6) as participant organizations within the network.

To ensure proper configuration, users must update the details of the founder organization in the Central Bank (CB) section and the participant organizations in the Financial Institution (FI) section. It is essential to maintain a fixed order for the participant organizations: FI1 corresponds to Participant 1, FI2 to Participant 2, and so on. The same details should be used to update the respective FI details accordingly.

1. Sign in to your Oracle Cloud Infrastructure account.

   Ensure you're in the correct compartment where you'll deploy the sample application.
2. In the Console, click the **Navigation** menu in the top-left corner.
3. Under **Developer Services**, select **Oracle Blockchain Platform**.
4. Click **Create Oracle Blockchain Platform**.
   1. **Instance/Display Name**: Must contain 1-15 characters, starting with an ASCII letter.
   2. **Description**: Enter an optional description for your instance.
   3. **Platform Role**: Choose **Create a new network** to create a founder organization. For a participant instance, choose **Join an existing network.**
   4. **Platform Version**: Choose **Hyperledger Fabric v2.5.x**.
   5. **Edition**: Select **Digital Assets**.
5. Review your settings and click **Create**.

   The instance can take approximately 15 minutes to create. You'll receive notification once it's complete.

Create Users and User Groups with Oracle Identity Cloud Service

The CBDC application supports 11 personas, and the corresponding 11 application roles have already been created in the Visual Builder package. These roles are necessary to define the permissions and access levels for each persona in the application.

For a complete list of the roles and their operations, see Wholesale CBDC Application Workflow.

Application roles in Visual Builder are created to:

- **Define Access Levels**: Each persona (example, Central Bank Admin, Participant User) has specific permissions and access requirements in the application. Application roles ensure that users only see and interact with the features relevant to their role.
- **Enable Role-Based Access Control (RBAC)**: By mapping IDCS groups to these roles, you can control who has access to what within the application.
- **Simplify User Management**: Instead of assigning permissions to individual users, you assign them to roles, and users inherit these permissions through their IDCS group membership.

Overview

The IDCS groups for 1 Central Bank (CB) and 6 Participants have already been mapped to these application roles in Visual Builder. This means you only need to create IDCS groups and add users to those groups as listed in the table below. The IDCS groups are already mapped to the corresponding application roles in Visual Builder. Once users are added to the groups, they will automatically get the correct access to the application.

You'll create the groups described in the table below, and add users to them. By creating the IDCS groups with the exact names provided and adding users to these groups, you can easily enable role-based access to the application. The mapping between IDCS groups and Visual Builder roles is already configured, so no further setup is required.

For additional informing on creating IDCS groups and managing users, see: Manage Oracle Identity Cloud Service Users and Manage Oracle Identity Cloud Service Groups.

Table 3-1 Application Roles and their IDCS Groups and Bank Names

S NO	Application Role	IDCS User Groups	Bank Name
1	CBDC_ADMINS	CB_CBDC_ADMINS	CB
2	CBDC_AUDITORS	CB_CBDC_AUDITORS	CB
3	CBDC_CREATORS	CB_CBDC_CREATORS	CB
4	CBDC_ESCROW	CB_CBDC_MANAGERS	CB
5	CBDC_ISSUERS	CB_CBDC_ISSUERS	CB
6	CBDC_RETIRERS	CB_CBDC_RETIRERS	CB
7	FI_ADMINS	FI1_CBDC_ADMINS (repeat this pattern for remaining participant orgs like <org>_CBDC_ADMINS)	FI1, FI2, FI3, FI4, FI5 and FI6
8	FI_CBDC_USERS	FI1_CBDC_USERS (repeat this pattern for remaining participant orgs like <org>_CBDC_USERS)	FI1, FI2, FI3, FI4, FI5 and FI6
9	FI_CBDC_OFFICERS	FI1_CBDC_OFFICERS (repeat this pattern for remaining participant orgs like <org>_CBDC_OFFICERS)	FI1, FI2, FI3, FI4, FI5 and FI6
10	FI_CBDC_MANAGERS	FI1_CBDC_MANAGERS (repeat this pattern for remaining participant orgs like <org>_CBDC_MANAGERS)	FI1, FI2, FI3, FI4, FI5 and FI6
11	FI_CBDC_AUDITORS	FI1_CBDC_AUDITORS (repeat this pattern for remaining participant orgs like <org>_CBDC_AUDITORS)	FI1, FI2, FI3, FI4, FI5 and FI6

Create Groups

1. Sign in to your Oracle Cloud Infrastructure account. Ensure you're in the correct compartment where you'll deploy the sample application.
2. In the Console, click the **Navigation** menu in the top-left corner. Click **Identity & Security**. Under **Identity** select **Domains**.
3. On the **Domains** page, click **Oracle Identity Cloud Service** to open the Domains Overview page.
4. Click **Groups**. Click **Create Group**.
   - **Name**: Enter a unique name for the group (example `CB_CBDC_ADMINS`).
   - **Description**: Provide a brief description of the group's purpose.
   - To allow users to request access to this group, select the option **User can request access**.

   Click **Finish**.

Create Users and Assign Them to Groups

1. On the Domains Overview page, click **Users**.
2. Click **Create User**.
   - **First Name**: Enter the user's first name.
   - **Last Name**: Enter the user's last name.
   - **User Name / Email**: Enter a valid email address or username for login.
   - **Email**: Enter the email address for communication and account activation.
3. On the Assign Group page, you will see a list of existing groups.
4. Select the checkbox next to each group you want to assign this user to. Ensure you select the appropriate group that aligns with their role (example CB_CBDC_ADMINS).
5. After selecting the desired groups, click **Finish** to complete user creation.

Verifying Users and Groups

1. After creating groups and adding users, return to the **Groups** section in the IDCS Console.
2. Verify that all created groups and added users are listed correctly.

Configure Oracle Blockchain Platform for the Wholesale CBDC Sample Application

After you've created all the required instances of Oracle Blockchain Platform and it's prerequisite products, you'll need to configure your users and roles, add any participant organizations to the founder organization, and enable the rich history database.

Configure IDCS Groups to Oracle Blockchain Platform Instances

For additional information on the built-in Oracle Blockchain Platform roles and how to assign users to them, see Set Up Users and Application Roles

For additional information on the wholesale CBDC sample application roles, see Wholesale CBDC Application Workflow.

In Oracle Blockchain Platform, the following application role assignments should be implemented:

- **Central Bank (CB) Groups**: Assign all Central Bank groups to the `REST_CLIENT` application role of CentralBank Oracle Blockchain Platform instance.
- **Financial Institution (FI) Groups**: Assign all Financial Institution groups to the `REST_CLIENT` application role as well of their respective organization.
- **Administrative Roles**: Assign the relevant administrative groups, such as `CB_CBDC_ADMIN` and `FI_CBDC_ADMINS`, to the `ADMIN` application role of their respective organization.

This configuration ensures that both Central Bank and Financial Institution groups have the necessary access through the `REST_CLIENT` role, while administrative privileges are appropriately managed through the `ADMIN` role.

Follow these steps to assign the groups to application roles.

1. Sign in to your Oracle Cloud Infrastructure account.
2. In the Console, click the **Navigation** menu in the top-left corner.
3. Under **Identity**, select **Domains**.

4. On the Domains page, click **Oracle Identity Cloud Service**.
5. In the Oracle Identity Cloud Service navigation menu, select **Oracle Cloud Services**. Locate the Oracle Blockchain Platform instance for which you want to assign group roles. Open that instance's Details page.
6. Go to the Application Roles tab to view the Oracle Identity Cloud Service application roles listed in the Resources navigator.
7. Select the role you want to assign, and expand the role. Click the More menu for the role and select **Assign Groups**.
8. In the Assign Groups dialog, select the groups you want to assign to the role, and click **Assign**.

Fetch `ClientId` and `ClientSecret`

The Oracle Blockchain Platform `ClientID` and `ClientSecret` will be utilized by Visual Builder. You'll need to update these parameters in Visual Builder later as described in Configure Visual Builder Backends.

To retrieve the client ID and client secret for an Oracle Blockchain Platform instance:

1. Sign in to your Oracle Cloud Infrastructure account. Ensure that you have sufficient administrative privileges to manage OAuth settings.
2. In the Console, click the **Navigation** menu in the top-left corner.
3. Under **Identity**, select **Domains**.
4. On the Domains page, click **Oracle Identity Cloud Service**.
5. In the Oracle Identity Cloud Service navigation menu, select **Oracle Cloud Services**. Locate the Oracle Blockchain Platform instance for which you want to fetch the client ID and client secret. Open that instance's Details page.
6. When you open the instance details page, the default tab displayed will be the OAuth Configuration tab. This tab contains essential information regarding OAuth settings for your Oracle Blockchain Platform instance.
   You will find fields labeled Client ID and Client Secret in the General Information section. These credentials are used to authenticate API requests to the Oracle Blockchain Platform.

Multiple Organization Configuration

If you're creating a complex Oracle Blockchain Platform network with multiple organizations, add participants to the founder. An overview of this process is provided in the Oracle Blockchain Platform Admin Guide: Add OBCS Participant Organizations to the Network.

Configure the Rich History Database for Oracle Blockchain Platform

For additional information about configuring and using the rich history database, see Create the Rich History Database .

Before completing these steps, you must have created an Autonomous Database instance as described in Provision Autonomous Database in order to enable the rich history database for Oracle Blochain Platform.

1. Open the Oracle Blockchain Platform console for the network instance.
2. Click on the **More Actions** menu in the top-right corner and select **Configure Rich History**.

3. Specify an Oracle database connection by providing 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.
4. Click **Save** to configure the rich history database.

Create Enrollments

You must manually create the enrollments for the users in `CB_CBDC_ADMINS` and `FI_CBDC_ADMINS` groups.

Enrollments in the Oracle Blockchain Platform's REST proxy provide a mechanism for users to interact with the blockchain without requiring an enrollment certificate, simplifying API calls by mapping application identities to blockchain members. To initiate any call to the chaincode, enrollments must be established within the Oracle Blockchain Platform.

The wholesale CBDC application manages the creation of enrollments for users during account setup; however, when a CBDC Admin user logs in for the first time, no enrollment exists. Consequently, if this user attempts to create an account, the `RestProxy` call will fail. Therefore, it is essential to create an enrollment for the CBDC ADMIN group.

When setting up accounts, enrollments are created for the respective organizations. For instance, when accounts for the Central Bank (CB) are established, corresponding enrollments will be generated. However, if any Financial Institution (FI) accounts are created through the CBDC Admin page, enrollments will not be automatically created for those FIs. In this wholesale CBDC workflow, it is necessary to set up FI admin accounts within the CB Admin page. Therefore, users must manually create enrollments for the FI Admin group as these enrollments will not be generated automatically due to the distinct organizational structure between the Central Bank and Financial Institutions.

Refer to Add Enrollments to a REST Proxy for comprehensive information on how to create the enrollments.

Fetch `RestproxyId`

The `RestproxyId` will be utilized by Visual Builder. You'll need to update this in Visual Builder later as described in Configure Visual Builder Backends.

To retrieve the `RestProxyId` for an Oracle Blockchain Platform instance, users must invoke the Oracle Blockchain Platform console APIs using an administrative user account. Authentication can be performed via either basic authentication or OAuth 2.0.

Authentication Method

For OAuth 2.0, refer to the documentation at OAuth 2.0 Access Token Based Authentication to implement the necessary authentication flow.

API Endpoint for Fetching `RestProxyId`

Use the Oracle Blockchain Platform console API to obtain the `RestProxyId` by sending a request to the following endpoint:

```
GET <obp_url>/console/admin/api/v2/nodes
```

This API call will return a response containing details about the REST proxies configured in your instance.

For more details about the console API refer to following documentation: Get Node List.

Response Details

In the response, locate the section labeled `RESTProxies`, where you will find the corresponding `RestProxyId`. The structure of the response will include various attributes associated with each REST proxy node.

```
"RESTProxies": [
        {
            "status": "up",
            "extInfo": [],
            "displayName": "restproxy",
            "RESTProxyId": "48021a2c-e62b-40bd-b7ca-71dda883521d-restproxy",
            "url": "https://fibank3-oabcs1-
hyd.blockchain.ocp.oraclecloud.com:7443/restproxy",
            "mspId": "FIBank3"
        }
    ]
```

Deploy the Chaincode for the Wholesale CBDC Sample Application

You can now create a channel and deploy the sample application chaincode to it.

1. Create a channel.
   a. Define a new channel with the necessary participants as per your organisational requirements.
   b. During the channel creation process, ensure **Enable Rich History** is selected. This option activates the rich history feature, which was previously configured.
2. Deploy the chaincode.

   Deploy the wholesale CBDC chaincode provided in the wholesale CBDC chaincode package to the newly created channel. For details on how to deploy chaincodes, refer to Typical Workflow to Deploy Chaincodes.
3. Use the wholesale CBDC Postman collection provided in the wholesale CBDC chaincode package to invoke the `init` API. This step completes the initialization process, ensuring the chaincode is ready for use.

   See Wholesale CBDC Chaincode Package.
4. Check the replication status.
   a. Go to the **Channels** tab.
   b. Locate the channel and click on the **More Actions** menu on the right side of the channel entry.
   c. Select **Rich History Status** to view the replication status.
      - If the status is `REPLICATING`, it indicates that the channel data is actively replicating to the rich history database.
      - If the replication status is `USER_DISABLED` perform the following actions:
        i. Restart the Peer: Restart the peer node associated with the channel.
        ii. Invoke a Transaction: Trigger any transaction on the channel to initiate data replication.
        iii. Recheck the replication status to confirm that the data is now replicating to the rich history database.

Note the channel name and chaincode name - you'll need to configure Visual Builder with these in order to work with the sample application.

Create Rich History Database Views

After you've created the rich history database for your Oracle Blockchain Platform instance, you can run the Database View Creation script to create the views in the rich history database and generate the ORDS endpoint.

The Oracle REST Data Services (ORDS) script is a Node.js application written in TypeScript, designed to expose ORDS endpoints for retrieving account transaction details from the rich history database. It creates a RESTful endpoint that allows users to fetch chaincode transaction data (for both TypeScript and Go chaincodes) from the rich history database. In addition, it provides essential credentials - such as the ORDS endpoint, `clientId`, `clientSecret`, and Bearer token - to authenticate and access the endpoint.

1. Download the Database View Creation script.
   a. Open the Oracle Blockchain Platform console.
   b. Go to the Digital Assets tab.
   c. In the left sidebar, go to the Wholesale CBDC application.
   d. Download the script from the **Oracle Database View Definitions for WholesaleCBDC** area.
2. After downloading the script, refer to Oracle Database View Definitions for Wholesale CBDC for details on how to execute it.

   When running the script, you will be prompted with a question: select TypeScript (TS) as the language, since the wholesale CBDC chaincode is written in TypeScript.
3. The script returns the ORDS endpoint, client ID and client secret.

```
{
  ORDSEndpoint: 'https://<base_URL>ords/<user_name|alias>/<resource_link>',
  clientId: '<clientId>',
  clientSecret: '<clientSecret>',
  bearerToken: {
    access_token: '<BearerToken>',
    token_type: 'bearer',
    expires_in: 3600
  }
}
```

The ORDS endpoint for accessing RESTful services follows this structure:

```
ORDS_REST_BASE_URL/ords/ALIAS_NAME/BASE_PATH/PATTERN
```

- ORDS_REST_BASE_URL: The base URL of the ORDS service.
- ALIAS_NAME: The database or schema alias configured in ORDS.
- BASE_PATH: The base path for the RESTful services.
- PATTERN: The specific endpoint pattern for the RESTful service.

To generate a bearer token for authentication, use the OAuth2 token endpoint:

```
ORDS_REST_BASE_URL/ords/ALIAS_NAME/oauth/token
```

The following parameters related to ORDS must be updated in the backend of Visual Builder to retrieve data from the rich history database. This will be completed later in Configure Visual Builder Backends.

- clientId
- clientSecret
- ORDS endpoint
- OAuth base URL: `ORDS_REST_BASE_URL/ords/<userName|aliasName>`

Import the Wholesale CBDC Sample Application into Visual Builder

After the prerequisites are complete, you can import the sample application into Oracle Visual Builder.

Prerequisites:

- Access to Oracle Visual Builder: Ensure you have the necessary permissions to access and modify the Visual Builder instance.

To import the sample application into Visual Builder, complete the following steps.

1. Download the Visual Builder sample application package from Oracle Blockchain Platform.
   a. Open the Oracle Blockchain Platform console.
   b. Click the **Digital Assets** tab.
   c. Select **Wholesale CBDC Application** from the left sidebar.
   d. Under **Sample wCBDC Web Application and Analytics Workbook**, select **Download sample VBCS UI and sample Analytics package**. This downloads the `WholesaleCBDCVBCS.zip` file, which contains the sample application.
   e. Extract the `WholesaleCBDCVBCS.zip`, which contains the `WholesaleCBDCAnalyticsPackage` folder and another file called `WholesaleCBDCVBCSPackage.zip`, which is the Visual Builder application archive file.
2. Sign in to Oracle Visual Builder.
3. Go to your Visual Applications Home page and click **Import**.
4. In the Import dialog box, click **Application from file**.
   - Drag the visual application archive file on your local system into the dialog box.
   - Alternatively, click the upload area in the dialog box and use the file browser to locate the archive on your local system.
5. Enter the `Application Display Name` and `Application ID`. Both fields are automatically populated based on the archive name, but you might want to modify the name and the ID to be unique in your identity domain.
6. Click **Import**.

When you importing the sample application into a new tenancy, you might see the following warning.

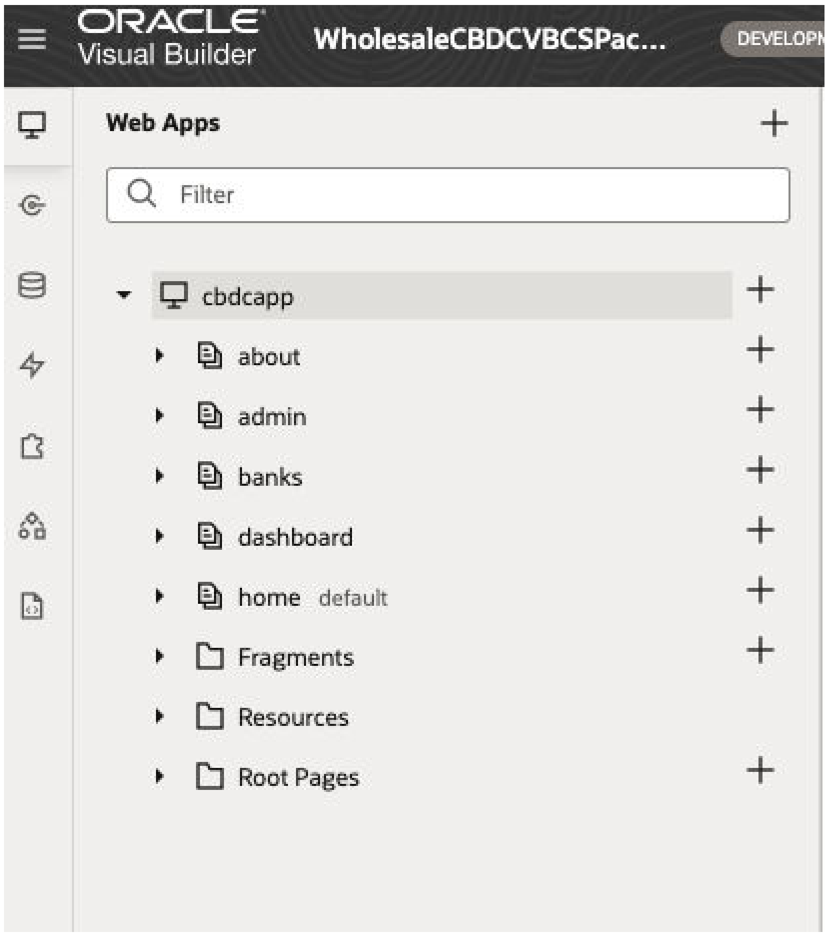
```
Import warning: applicationRoleImport
```

This occurs if you did not create all of the users and user groups as described in Create Users and User Groups with Oracle Identity Cloud Service. For example, if you created users and groups for only the central bank and two financial institutions, you will see warnings related to the mappings for the four remaining financial institutions. If Oracle Identity Cloud Service (IDCS) groups that are mapped to application and user roles in the original package do not exist in the target tenancy, the import process removes those mappings. You can safely ignore these warnings.

To add another financial institution after you import the application, you must complete the following steps to manually map IDCS user groups to the corresponding user roles.

1. Log on to the OCI console and then click **Identity & Security** > **Oracle Identity Cloud Service (IDCS)**.
2. Under **Groups**, if any of the required IDCS groups are missing, create them.
3. Navigate to the **Users** area in IDCS, and then add users to the IDCS groups based on their roles.
4. Click **Developer Services** > **Visual Builder**, and then open the `cbdcapp` application.
5. In Oracle Visual Builder, click **Settings** > **User Roles**.
6. For each user role, select the role, click **Edit Role Mapping**, and then select the appropriate IDCS group.
7. Save the changes.

After importing the package completes, you can see the wholesale CBDC application on the Visual Applications home page. Click **Web Applications** in the Navigator to open the Web Apps pane. You will see the `cbdcapp` application as shown in the following screen capture:



Configure Visual Builder for the Wholesale CBDC Sample Application

After you've configured all the other products associated with Oracle Blockchain Platform, you can configure Oracle Visual Builder to connect to them.

Overview

The following details must be updated in the Oracle Visual Builder configuration. You get these details when you configure Oracle Blockchain Platform as described in Configure Oracle Blockchain Platform for the Wholesale CBDC Sample Application.

- REST proxy ID, instance URL and MSP IDs of the Oracle Blockchain Platform instance.
  See: Fetch `RestproxyId`
- Client ID and client secret of the Oracle Blockchain Platform instance.
  See: Fetch `ClientId` and `ClientSecret`.
- Oracle REST Data Services (ORDS) endpoint, client ID, client secret, and OAuth base URL.
  See: Create Rich History Database Views.
- Channel name and chaincode name.
  See: Deploy the Chaincode for the Wholesale CBDC Sample Application.

Update Global Variables

1. On the Visual Builder Visual Applications home page, click **Web Applications** in the Navigator to open the Web Apps pane.
2. Click the `cbdcapp` application. The app editor opens.
3. Select **Variables**.
4. In the **Global Variables** section, locate the `configuration` variable object and update the following parameters in the object:

Parameter Name	Description	Default Value
`chaincodeName`	The `chaincodeName` used in the Oracle Blockchain Platform network.	`WholesaleCBDC`
`ordsUrl`	The ORDS endpoint URL, which is displayed in the output of the database view creation script.	`https://g53630e55ee33f4-test.xyz.abc.oraclecloudapps.com/ords/obp/cbdc/accountTrxDetails`

Configure Visual Builder Backends

A backend service connection in Visual Builder is a way to establish communication between your visual application and external systems by providing essential details such as connection information, properties, and REST API endpoints needed to access those systems. You need to update the backends with Oracle Blockchain Platform and rich history database details.

For more information, refer to What Are Backends?.

The following backends need to be updated:

- REST proxy calls
- Enrollments

- Rich history database configuration database

REST proxy calls

There is one backend and six service connections related to the REST proxy. These components are used to make REST proxy calls to the Oracle Blockchain Platform instance.

Backend
`CentralBank`

Service connections
`participantBank_FI1`

`participantBank_FI2`

`participantBank_FI3`

`participantBank_FI4`

`participantBank_FI5`

`participantBank_FI6`

To update the backend server details:

1. On the Visual Builder Visual Applications home page, click **Services** in the Navigator to open the Services pane. Click **Backends** to see a list of backends.
2. Select the backend you want to update. Click **Servers**.
3. Click the Edit icon next to the default server.
4. Update the following in the Server Details:
   - Instance URL: Replace the default URL with the Oracle Blockchain Platform instance URL by modifying the base URL portion of the default URL (`test-xyz-abc.blockchain.ocp.oraclecloud.com`). For example:

     ```
     https://centralbank-oabcs1-hyd.blockchain.ocp.oraclecloud.com:7443/
     restproxy/api/v2/channels/{channelName}
     ```

   - Server variables: Change `channelName` to reflect the name of the channel where the chaincode is deployed.
   - Authentication for logged-in users: OAuth 2.0 User Assertion is used for authentication. Modify the URL to match the Oracle Blockchain Platform REST proxy URL. For example:

     ```
     https://centralbank-oabcs1-hyd.blockchain.ocp.oraclecloud.com:7443/
     restproxy
     ```

   - Client ID and client secret: Click the Edit icon next to **ClientID**. Update the `Client ID` and `Client Secret` fields to match your Oracle Blockchain Platform ID and secret.
5. Repeat these steps to update the service connections of each of the six participants by selecting them in the Service Connections View in the Services pane.

Enrollments

There are seven backends related to Oracle Blockchain Platform enrollments. These backends are used to create the enrollments to the Oracle Blockchain Platform instance.

Backends

`CustomEnrollementCB`

`CustomEnrollementFI1`

`CustomEnrollementFI2`

`CustomEnrollementFI3`

`CustomEnrollementFI4`

`CustomEnrollementFI5`

`CustomEnrollementFI6`

To update the backend server details:

1. On the Visual Builder Visual Applications home page, click **Services** in the Navigator to open the Services pane. Click **Backends** to see a list of backends.
2. Select the backend you want to update. Click **Servers**.
3. Click the Edit icon next to the default server.
4. Update the following in the Server Details:
   - Instance URL: Replace the default URL with the Oracle Blockchain Platform instance URL by modifying the base URL portion of the default URL (`test-xyz-abc.blockchain.ocp.oraclecloud.com`). For example:

     ```
     https://centralbank-oabcs1-hyd.blockchain.ocp.oraclecloud.com:7443/
     console/admin/api/v2/nodes/restproxies/{restProxyId}/enrollments
     ```

   - Server variables: Change `restProxyId` to reflect the actual REST proxy ID for your Oracle Blockchain Platform instance.
   - Authentication for logged-in users: OAuth 2.0 User Assertion is used for authentication. Modify the URL to match the Oracle Blockchain Platform REST proxy URL. For example:

     ```
     https://centralbank-oabcs1-hyd.blockchain.ocp.oraclecloud.com:7443/
     restproxy
     ```

   - Client ID and client secret: Click the Edit icon next to **ClientID**. Update the `Client ID` and `Client Secret` fields to match your Oracle Blockchain Platform ID and secret.
5. Repeat these steps to update the backends of each of the six participants by selecting them in the Backends View in the Services pane.

Rich history database configuration database

There is one backend related to the rich history database. This backend is used to fetch the data from the rich history database.

Backend
`RichHistoryDBConfiguration DB`

To update the backend server details:

1. On the Visual Builder Visual Applications home page, click **Services** in the Navigator to open the Services pane. Click **Backends** to see a list of backends.
2. Select the `RichHistoryDBConfiguration DB` backend. Click **Servers**.
3. Click the Edit icon next to the default server.
4. Update the following in the Server Details:
   - Instance URL: Replace the default URL with `OAuth Base URL(ORDS_REST_BASE_URL/ords/<userName|aliasName>)`. For example the default URL:

     ```
     https://g53630e55ee33f4-abc.def.gh-xyz-1.oraclecloudapps.com/ords/
     aliasName
     ```

     would be replaced by:

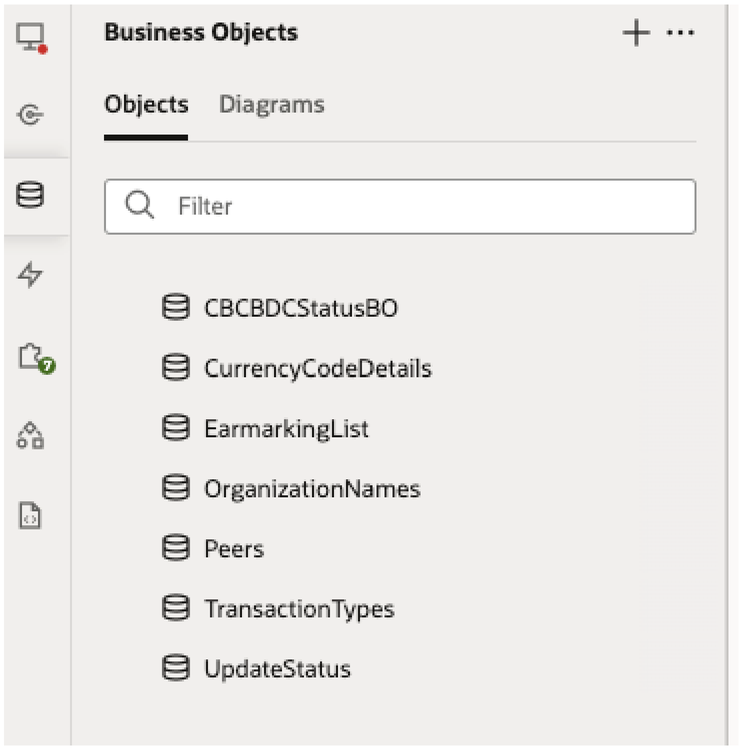
     ```
     https://g53630e55ee33f4-rhjkdb.adb.ap-sydney-1.oraclecloudapps.com/ords/
     vbcstest
     ```

   - Authentication for logged-in users: Basic is used for authentication.
   - Client ID and client secret: Click the Edit icon next to **Username**. Update the user name with the `Client ID` and the password with the `Client Secret` to match your ORDS endpoint ID and secret.

Configure Visual Builder Business Objects

A business object is a resource, such as an invoice or purchase order, similar to a database table; it has fields that hold the data for your application. Like a database table, a business object provides the structure for data. Business objects are stored in a database. The apps in your visual application and other clients access the business objects via their REST endpoints.

On the Visual Builder Visual Applications home page, click **Business Objects** in the Navigator to open the Business Objects pane. The wholesale CBDC objects should be listed.



CBCBDCStatusBO
CurrencyCodeDetails
EarmarkingList
OrganizationNames
Peers
TransactionTypes
UpdateStatus

You can select any business object and go to its Data tab to update or modify its data.

Organization Names

This business objects has four fields in the Data tab and helps connect the Oracle Blockchain Platform instances with the right backend for each bank. For example, if a user enters `Bank1` details in the `FI1` system, the application needs to link `FI1` with `Bank1`. That way, whenever a user associated with `Bank1` logs in, the system knows to pull data from the `FI1` backend.

In this business object, you need to update the `BankName` and `mspId` columns for the respective `orgNames`. The updates should correspond to the number of participants (FIs) in the network. For example, if there are `2 FIs (FI1, FI2)` in the network, you must update the `BankName` and `mspId` columns for these two FIs. For the remaining FIs (`FI3, FI4, FI5, FI6`), the `BankName` and `mspId` columns should be left `empty`.

Field Name	Description	Default Values	Update Needed (Yes/No)	Update Requirement
BankName	This is the mapping used between Oracle Blockchain Platform instance names and `BankNames` used in the CBDC Application. For example, if you created the Oracle Blockchain Platform instance with the name `CentralBank` but you want `BankName` to be `BSP` in the application , this mapping creates the mapping between the Oracle Blockchain Platform instance and `BankName`.	`CentralBank, Bank1, Bank2, Bank3, Bank4, Bank5` and `Bank6`	Yes	Required for active FIs; leave empty for unused FIs.

Field Name	Description	Default Values	Update Needed (Yes/No)	Update Requirement
mspId	This column represents the actual Oracle Blockchain Platform instance MSP ID. You must update these values if Oracle Blockchain Platform instances are created with names other than default values mentioned below.	`CentralBank`, `Bank1`, `Bank2`, `Bank3`, `Bank4`, `Bank5` and `Bank6`	Yes	Required for active FIs; leave empty for unused FIs.
OrgName	This column represents `orgNames` that map with the backend. `CB` represents the founder and `FI#` represents the participants.	`CB`, `FI1`, `FI2`, `FI3`, `FI4`, `FI5` and `FI6`	No	---

Peers

This business object is designed to store information about the peers in all Oracle Blockchain Platform instances within the network. To input the peer URL details into the business object, you should enter the corresponding peer URL values in the **peerURL** column along with their respective Oracle Blockchain Platform Membership Service Provider IDs (`mspId`).

For example, if there is one Central Bank (`CB`) and two Financial Institutions (FI#), each with two peers, you must enter a total of six peer URLs along with their associated `mspId`.

Field Name	Description
orgName	Oracle Blockchain Platform instance `mspId`.
peerURL	The `peerURL` value of Oracle Blockchain Platform instances.

Note:

The peers included in this Business Interface should be added to the common channel that has been created.

Configure Oracle Analytics Cloud for Oracle Visual Builder

The Oracle Analytics Cloud instance and the Oracle Visual Builder instance must reside in the same OCI tenancy.

The Oracle Analytics Cloud instance and the Oracle Visual Builder instance must also use the same Oracle Identity Cloud Service for identity management. Users who access the Oracle Visual Builder application must have the necessary privileges for both Oracle Analytics Cloud and Oracle Visual Builder.

Before you complete the following steps, configure the wholesale CBDC analytics package. For more information, see Wholesale CBDC Sample Analytics Package.

1. Get the Oracle Visual Builder domain URL.
   a. On the OCI console, click **Developer Services** > **Visual Builder**.
   b. Select your Visual Builder instance, and then click **Service Homepage**. The Visual Applications interface page opens. Copy the domain name from your browser's address bar.

Example domain name: `Example: https://wcbdc-vb-oabcs1.builder.ap-hyderabad-1.ocp.example.com`

2. Add the domain as a safe domain in Oracle Analytics Cloud.
   a. Log in to your Oracle Analytics Cloud instance, open the Oracle Analytics Cloud console, and then click **Navigator Menu**.
   b. Under Configuration and Settings, click **Console** > **Safe Domains**.
   c. Click **Add Domain**, and then paste the Visual Builder domain URL that you copied previously. The domain is added automatically when you click anywhere else on the display.
   d. Select the **Allow Frames** and **Embedding** check boxes associated with the domain that you just added.
3. Get the Oracle Analytics Cloud host URL.
   a. Log in to your Oracle Analytics Cloud instance.
   b. Copy the Oracle Analytics Cloud host URL from your browser's address bar. Copy the fully-qualified domain name up to `.com`, including `https://`.
   Example host URL: `https://<your-oac-instance>.analytics.ocp.example.com`
4. Add the Oracle Analytics Cloud host URL as an allowed origin in Oracle Visual Builder.
   a. Log in to your Oracle Visual Builder instance, and then click **Service Homepage**.
   b. Click **Navigation Menu**, then click **Settings**. The Tenant Settings page is displayed.
   c. In the Allowed Origins section, click **+ New Origin**, and then paste the Analytics Cloud domain URL that you copied previously into the **Origin Address** field.
5. Update the configuration variables in the Visual Builder application.
   a. Open the `cbdcapp` application in the app editor, and then select **Variables**.
   b. In the Global Variables section, update the following parameters in the `configuration variable` object.

      `OACHost`
      The Oracle Analytics Cloud instance URL, which you copied previously.

      `OACCBDCProjectPath`
      The project path for the central bank workbook in Analytics Cloud.

      `OACFIProjectPath`
      The project path for the financial institution workbook in Analytics Cloud.
   c. To get the project path of a workbook, click **Navigator** > **Catalog** on the Analytics Cloud home page.
   d. Click the **Actions** menu for the workbook, and then select **Inspect**. The Inspect window is displayed.
   e. Click the **General** tab, and then click **Copy** that is associated with the **Object ID**.
6. Edit the HTML in Visual Builder.
   a. Open the `cbdcapp` application in the app editor, and then navigate to the HTML section.

b. Update the Analytics Cloud host URL on lines 30 and 31 to match your Analytics Cloud instance, as shown in the following code snippet. Do not change the subdirectory structure of the URL, only the domain portion up to `.com`.

```
<script src="https://abc-xyz-ia.analytics.ocp.example.com/public/dv/v1/
embedding/jet/embedding.js" type="application/javascript"></script>
```

Customize the Wholesale CBDC Application

You can change the default token and the transaction types that are used by the wholesale CBDC application.

CBDC Token

The default token in the application is USD. To change the default token, complete the following steps.

1. In the global variables section, update the following parameters in the configuration variable object.

   `currencySymbol`
   The symbol that represents the currency. The default value is the dollar sign ($).

   tokenId
   The token ID that represents the currency. The default value is `USD`.

2. In the `CurrencyCodeDetails` business object, update the following parameters.

   currencyDesc
   The description of the currency. The default value is `US Dollar`.

   currencyName
   The token ID that represents the currency. The default value is `USD`.

3. Update currency symbol in the CSS files.

   a. Navigate to **Web Apps** and then expand the **Resources** section.

   b. Open the `Resources/css/app.css` file.

   c. Update the value of the *token-content* variable on line 10 of the file. The default value is the dollar sign ($), as shown in the following example.

   ```
   :root {
       --token-content: "$";
   }
   ```

Transaction Types

The `TransactionTypes` business object maps the transaction types in the chaincode to the transaction types in the application. For example, the `REQUEST_MINT` transaction in the chaincode is mapped to the `Creation Requested` transaction in the application. You can update the value fields in this business object to change the mappings, as shown in the following table.

TransactionType	value	Description	Use Case
`CREDIT`	Credit	Tokens are credited to the specified account.	A CBDC Creator successfully requests minting tokens and the system credits the tokens to their account.
`DEBIT`	Debit	Tokens are debited from the specified account.	A CBDC Creator transfers tokens to the CBDC Issuer.
`REQUEST_MINT`	Creation Requested	A request to mint tokens is submitted for approval.	A CBDC Creator submits a request to mint tokens.
`APPROVE_MINT`	Creation Approved	A mint request is approved	A CBDC Manager approves a mint request.
`REJECT_MINT`	Creation Rejected	A mint request is rejected.	A CBDC Manager rejects a mint request.
`REQUEST_BURN`	Retirement Requested	A request to burn tokens is submitted for approval.	A CBDC Retirer submits a request to burn tokens.
`APPROVE_BURN`	Retirement Approved	A burn request is approved.	A CBDC Manager approves a burn request.
`REJECT_BURN`	Retirement Rejected	A burn request is rejected.	A CBDC Manager rejects a burn request.
`EXECUTEHOLD`	Issuance Approved	An issuance request is approved.	A CBDC Manager approves an issuance request.
`RELEASEHOLD`	Issuance Rejected	An issuance request is rejected.	A CBDC Manager rejects an issuance request.
`ONHOLD`	Request Issuance	A request to issue tokens is submitted for approval.	A CBDC Issuer submits a request to issue tokens to an FI Officer.

Stage the Wholesale CBDC Application

You can stage the application in Oracle Visual Builder to test and review it before deploying it to a production environment.

To stage the application, complete the following steps.

1. Log in to Oracle Visual Builder.
2. On the home page, click the icon of the application.
3. Click the **Application Options** icon, and then select **Stage**. The Stage Application window opens.
4. The first time you stage the application, select **Populate Stage with Development Data**, which copies all of the business object data from the development environment to the stage environment. If you stage the application again, select **Keep Existing Data in Stage**, which does not copy the business object data.
5. Access the staged application.
   a. After the application is staged, navigate back to the home page.
   b. On the application dashboard, locate the staged application. The status is shown as `Stage` next to the application name (for example, `WholesaleCBDCVBCSPackage`).

   c. Click the drop-down menu under `Stage` and select the application name (for example, `cbdcapp`).
   d. Copy the link or open the staged application in a new browser tab or window to test and review the application.

If you make changes to the application, repeat the previous steps.

If you make changes to the business objects after staging the application, complete the following steps before you stage the application again.

1. Verify that the business object data in the development environment matches the data in the stage environment.
   a. Log on to Oracle Visual Builder.
   b. Navigate to the application and then select **Business Objects**.
   c. Click the **Data** tab that is associated with a business object.
   d. Select **Development** from the list to view the data in the development environment.
   e. Select **Staging** from the list to view the data in the stage environment.
2. If the data does not match, export the data from the stage environment in `.csv` format. Import this data into the development environment.
3. After you confirm that the data in the development environment matches the data in the stage environment, follow the previous steps to stage the application.

After staging the application and verifying that it functions correctly, you can publish the application to production. You cannot change the published version of an application. To make changes to the application, you must create a new version.

Troubleshoot the Wholesale CBDC Sample Application

You can manually enter account and token data if you lose data during the staging process.

The wholesale CBDC sample application stores account and token data in business objects in Oracle Visual Builder. Specifically, account details are stored in the `CBCBDCStatusBO` business object and token details are stored in the `EarmarkingList` business object. When you stage the application for the first time, these business objects are empty. As you interact with the application by initializing tokens and creating accounts, data is saved to these business objects.

If you stage the application again and do not select **Keep Existing Data in Stage**, the business objects are reset. This can lead to data mismatches between the business objects and the ledger, which in turn can cause application malfunctions.

You can prevent this issue by backing up all stage data before you stage the application again, and by using the Data Manager tool in Oracle Visual Builder to export and import business object data between environments.

If you mistakenly select **Stage Application with a Clean Database** or **Replace Stage Data with Development Data** when re-staging the application, you must manually re-enter the account and token details by completing the following steps.

1. Use the `getAllActiveAccounts` and `getAllSuspendedAccounts` endpoints in the wholesale CBDC Postman collection to get information about all accounts in the network.
2. Populate the `CBCBDCStatusBO` business object as described in the following tables.

CBCBDCStatusBO Field	Source Field	Notes
`bankAccountID`	`account_id`	
`bankName`	`org_id`	
`bankStatus`		`Active` if the information comes from the `getAllActiveAccounts` endpoint, `InActive` if the information comes from the `getAllSuspendedAccounts` endpoint.
`bankTokenID`	`token_id`	
`bankUserID`	`user_id`	
`bankUserRole`	`role_name`, `non_account_role_name`	See the following table for role mapping information.
`groupName`	`application_groups`	
`userRole`		See the following table for role mapping information.

Condition	Value
`role_name` is null AND `non_account_role_name` is empty	NO
`role_name` is null AND `non_account_role_name` = `token_admin`	Token Admin
`role_name` is null AND `non_account_role_name` = `org_admin`	Org Admin
`role_name` is null AND `non_account_role_name` = `token_auditor`	Token Auditor
`role_name` is null AND `non_account_role_name` = `org_auditor`	Org Auditor
`non_account_role_name` is empty AND `role_name` is not null	The value of `role_name`

3. Leave all other fields in the `CBCBDCStatusBO` business object empty.
4. Use the `CurrencyCodeDetails` business object to get information about tokens and populate the `EarmarkingList` business object with this information, as described in the following table. You must re-enter information for all tokens that were previously initialized.

EarmarkingList field	Source	Description	Example
`currencyCode`	`currencyName` in `CurrencyCodeDetails`	Token ID used by the wholesale CBDC application	USD
`currencyString`	`currencyDesc` in `CurrencyCodeDetails`	Token description used by the wholesale CBDC application	US Dollar
`earmarkingString`		Purpose field entered when the token was initialized	[user defined]

Wholesale CBDC Application Workflow

After you install and configure the sample wholesale CBDC application, you can use it in scenarios where a central bank and other financial institutions interact in an interbank market.

The sample application supports eleven roles, or personas. Each role has a different interface and set of operations that support the entire workflow of token management in the wholesale CBDC scenario.

Central Bank Roles

- `CBDC Admin`: Manages the entire CBDC system.
- `CBDC Creator`: Creates CBDC tokens. After a creation request is submitted, it is sent to the CBDC approver, who either approves or rejects the request. After the minting request is approved, tokens are credited to the CBDC creator's account. The creator can then transfer these tokens to a CBDC issuer.
- `CBDC Approver/CBDC Escrow`: Approves or rejects requests for minting, burning, and holding CBDC tokens. Hold request approvals are used for transfers between organizations.
- `CBDC Issuer`: Transfers CBDC tokens to financial institution officers or to the CBDC retirer for burning. Users in this role receive tokens from the CBDC creator and financial institution officers. The CBDC approver must approve any transfers between organizations.
- `CBDC Auditor`: Has read-only access to all organizational data in the system.
- `CBDC Retirer`: Burns CBDC tokens. This role receives tokens from the CBDC issuer. After a burn request is submitted, it is sent to the CBDC approver.

Financial Institution Roles

- `FI Admin`: Manages their specific organization.
- `FI Officer`: Receives tokens from the CBDC issuer. They can transfer these tokens to users in any organization or return tokens to the CBDC issuer. All transfers between organizations must be approved by the financial institution approver.
- `FI User`: Receives tokens from the officers of their respective organizations. They can transfer these tokens to users and officers in any organization. All transfers between organizationsIt is essential to note that any inter-organizational transfers must be approved by the financial institution approver.
- `FI Manager/FI Approver`: Approves or rejects hold requests for their specific organization. Hold requests and approvals are used for all transfers between or within organizations.
- `FI Auditor`: Has read-only access to data that is specific to their organization.

Onboarding

After you install, configure, and stage the application, complete the following steps to access the application.

1. After the application is staged, navigate back to the home page.
2. On the application dashboard, locate the staged application. The status is shown as `Stage` next to the application name (for example, `WholesaleCBDCVBCSPackage`).

3. Click the drop-down menu under `Stage` and select the application name (for example, `cbdcapp`).
4. Copy the link or open the staged application in a new browser tab or window to test and review the application.

When a user attempts to log in to the application, the system checks that the user has an account and the appropriate role. If the account does not exist or if the required role is absent, the following error is displayed.

```
Invalid Account. Please Contact Admin
```

The interface that is displayed after a user logs in depends on their role.

The first time that any user attempts to log in to the application, no user accounts have been created yet. Only users from the `CBDC_ADMINS` group who also have the `tokenAdmin` role assigned to them can log in. All other login attempts will fail. The following section includes more information about the exception for the `CBDC_ADMINS` persona.

`CBDC_ADMINS` Exception

The login process is different for users with the `CBDC_ADMINS` persona. Users in the `CBDC_ADMINS` group can log in even if their account has not yet been created, but these users must have the `tokenAdmin` role.

When you deploy the chaincode, ensure that users in the `CBDC_ADMINS` group have the `tokenAdmin` role. The parameters that are passed during chaincode initialization must include the CBDC admin users as having the `tokenAdmin` role. This allows the `CBDC_ADMINS` to log in to the application for the first time to create the other user accounts.

If you log in to the application as a user in the `CBDC_ADMINS` group, and that user was not included as an initialization parameter when the chaincode was deployed, you must manually assign the `tokenAdmin` role to the user. You can manually assign the `tokenAdmin` role by using a Postman collection.

Application Workflow

The following steps show the actions of the various roles in a complete application workflow. The first seven steps must be completed to use the application.

1. The CBDC Admin logs in.
2. The CBDC Admin initializes the CBDC token.
3. The CBDC Admin creates their own bank account and then reloads the home page to see the updated network details.
4. The CBDC Admin creates bank accounts for all CBDC personas, as shown in the following table.

Application Group	Role
`CBDC_ADMINS`	Token Admin
`CBDC_AUDITORS`	Token Auditor
`CBDC_CREATORS`	Minter
`CBDC_ESCROW`	Escrow
`CBDC_ISSUERS`	none
`CBDC_RETIRERS`	Burner

5. The CBDC Admin creates FI Admin bank accounts, which have the Org Admin role.
6. The FI Admin logs in and creates accounts for users of their financial institution.
7. The CBDC Admin assigns roles to the new financial institution users, as shown in the following table.

Application Group	Role
`FI_ADMINS`	Org Admin
`FI_CBDC_USERS`	none
`FI_CBDC_OFFICERS`	none
`FI_CBDC_MANAGERS`	Escrow
`FI_CBDC_AUDITORS`	Org Auditor

8. The CBDC Creator logs in and requests that tokens be minted.
9. The CBDC Approver logs in and approves or rejects the minting request. If the request is approved, the tokens are credited to the CBDC Creator.
10. The CBDC Creator transfers the tokens to the CBDC Issuer.
11. The CBDC Issuer logs in and transfers the tokens to an FI Officer. If the transfer is approved by the CBDC Approver, the tokens are transferred. Alternately, the CBDC Issuer can transfer the tokens to the CBDC Retirer for burning.
12. The CBDC Auditor logs in, selects auditing policies, and reviews the relevant transaction data.
13. The CBDC Retirer logs in and requests that tokens be retired. If the CBDC Approver approves, the tokens are burned.
14. The CBDC Approver logs in and approves or rejects the request to issue tokens. If the transfer is approved, tokens are credited to the FI Officer, who can then transfer them to FI Users.
15. The FI Officer logs in and transfers tokens to FI Users, other FI Officers, or back to the central bank. All transfers require the approval of the FI Manager.
16. The FI Manager logs in and approves or rejects transfer requests.
17. FI Users log in and transfer tokens to other FI Users at any organization.
18. The FI Auditor logs in, selects auditing policies, and reviews the relevant transaction data.

CBDC Admin

The CBDC Admin manages the entire wholesale CBDC system.

A CBDC Admin user belongs to the `CBDC_ADMINS` group. The CBDC Admin initiates the token system and creates and manages accounts. The CBDC Admin can navigate to see and manage all organizations (banks) and all users in the system. Comprehensive details about the network and about users who are logged in are displayed, as well as filterable transaction history.

In the Accounts Management pane, all banks are displayed. You can navigate to each bank to suspend or activate user accounts, add or update roles, and edit user account details.

The Transaction History pane shows a list of filterable transactions. You can click any transaction to open the Transaction Details window.

The following tasks are typical ones that a CBDC Admin completes.

Initialize CBDC Token

Initializing the CBDC token is the first task that the CBDC Admin must complete.

1. Click **Initialize CBDC Token**. The Initialize CBDC Token window opens.
2. Select the token from the **Currency** list, and then optionally enter a **Purpose**. Separate multiple purposes with commas (,). These purposes can be selected as options during token transfers. The **Currency Description** field automatically populates based on the selected token, and cannot be edited.
3. Click **Initialize CBDC Token** to complete the process.

Create Bank Account

Complete the following steps to create bank accounts and add roles to accounts.

1. Click **Create Bank Account**. The Create Bank Account window opens.
2. Enter the account information, as described in the following table.

Field	Type	Description
Token ID	text, read-only	The ID of the token that was initialized for the system.
Bank Name	list	A list of all banks in the network.
Blockchain Network Member ID	text, read-only	The organization ID of the bank, which is determined by the selected **Bank Name**.
Bank User ID	text	A user ID for the account.
Role	list	A list of supported roles. The default is `NO`.
Group	list	A list of application groups.
Daily Maximum Amount	number	The maximum amount of tokens that can be transacted per day.
Daily Maximum Transactions	number	The maximum number of transactions allowed per day.

3. Click **Create Account**. The account details are displayed.

For central bank users, an enrollment is also created from the user ID during account creation. The enrollment ID can include only alphanumeric characters, hyphens (-), and underscores (_). All other characters are replaced by underscores.

Update Enrollments

1. Click **Update Enrollment**. The Update Enrollment window opens and all enrollments associated with a particular organization are displayed.
2. Select an enrollment ID and then click **Get User IDs**. All users that are associated with the enrollment ID are displayed.
3. To add a user to an enrollment, click **Add User**.
4. To delete a user from an enrollment, click **Delete** for that user.

CBDC Creator

The CBDC Creator mints tokens in the CBDC system and transfers them to issuers.

A CBDC Creator user belongs to the `CBDC_CREATORS` group and has the `Minter` role. The CBDC Creator can navigate and see their own filterable transaction history.

The following tasks are typical ones that a CBDC Creator completes.

Create Tokens

1. Click **Request Mint Tokens**. The Request Mint Tokens window opens.
2. Enter a **CBDC Quantity** and a **Memo**. All other fields are automatically populated and are read-only.
3. Click **Request Mint**. The transaction runs and is displayed in the Transaction History pane.

Issue Tokens

1. In the User Details pane, click **Issue**. The Transfer <TokenID> CBDC window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

CBDC Approver

The CBDC Approver approves or rejects requetst to mint, burn, and transfer tokens in the CBDC system.

A CBDC Approver user belongs to the `CBDC_ESCROW` group and has the `Notary` role. There can be only one CBDC Approver account but multiple users can share the account.

Comprehensive details about all requests from all banks are displayed for the CBDC Approver. The Approve / Reject CBDC Transfers to Member Banks pane shows all requests for three categories: **Issuance**, **Creation**, and **Retirement**. The CBDC Approver can select some or all requests to approve or reject them. The History page displays all requests that have been approved or rejected by the CBDC Approver.

CBDC Issuer

The CBDC Issuer transfers tokens to financial institution officers or to the CBDC Retirer.

A CBDC Issuer user belongs to the `CBDC_ISSUERS` group. The CBDC Issuer can navigate and see their own filterable transaction history.

The following tasks are typical ones that a CBDC Issuer completes.

Transfer Tokens to CBDC Retirer

1. In the Users pane, expand the central bank and then click **Retirement**. The Transfer <TokenID> CBDC window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

Transfer Tokens to Financial Institution Officers

1. In the Users pane, expand a financial institution and then click **Issue**. The Transfer <TokenID> CBDC to <FI Bank Name> window opens.

2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

CBDC Auditor

The CBDC Auditor applies auditing policies and reviews transactions that meet the specified policy conditions.

A CBDC Auditor user belongs to the `CBDC_AUDITORS` group and has the `Token Auditor` role. The CBDC Auditor can navigate and see filterable transaction history for all accounts in the network.

The following tasks are typical ones that a CBDC Auditor completes.

Apply Policies

1. In the Auditing Policies pane, select the policies to apply, as described in the following table.

Policy	Information
Single Transfer Amount	Select a symbol from the list and then enter an amount to audit single transfers.
Number of Transfers Exceeding	Enter a number of transfers and then select a period of time from the list to audit the number of transfers in that period.
Daily Transfer Total Amount Exceeding	Enter a value to audit the total daily transfer amount.
Any Transfers to Specific Account	Select a recipient to audit.
Any Transfers from Specific Accounts	Select a sender to audit.

2. Click **Apply**. The policies are applied and relevant transactions are displayed in the Transaction History pane.

CBDC Retirer

The CBDC Retirer burns CBDC tokens and transfers tokens to CBDC Issuers.

A CBDC Retirer user belongs to the `CBDC_RETIRERS` group and has the `Burner` role. The CBDC Retirer can navigate and see their own filterable transaction history.

The following tasks are typical ones that a CBDC Retirer completes.

Transfer Tokens to CBDC Issuers

1. In the User Details pane, click **Transfer**. The Transfer <TokenID> CBDC window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

FI Admin

The FI Admin manages a specific financial institution.

An FI Admin user belongs to the `FI_ADMINS` group and has the `Org Admin` role. The FI Admin creates and manages accounts for a specific financial institution. The FI Admin can navigate to see and manage all users in their financial insitution.

In the Accounts Management pane, all users for the specific bank are displayed. You can suspend or activate user accounts and edit user account details.

The following tasks are typical ones that a FI Admin completes.

Create Bank Account

Complete the following steps to create bank accounts.

1. Click **Create Bank Account**. The Create Bank Account window opens.
2. Enter the account information, as described in the following table.

Field	Type	Description
Token ID	text, read-only	The ID of the token that was initialized for the system.
Blockchain Network Member ID	text, read-only	The organization ID of the bank.
Bank User ID	text	A user ID for the account.
Group	list	A list of application groups.
Daily Maximum Amount	number	The maximum amount of tokens that can be transacted per day.
Daily Maximum Transactions	number	The maximum number of transactions allowed per day.

3. Click **Create Account**. The account details are displayed.

An enrollment is also created from the user ID during account creation. The enrollment ID can include only alphanumeric characters, hyphens (-), and underscores (_). All other characters are replaced by underscores.

Update Enrollments

1. Click **Update Enrollment**. The Update Enrollment window opens and all enrollments associated with a particular organization are displayed.
2. Select an enrollment ID and then click **Get User IDs**. All users that are associated with the enrollment ID are displayed.
3. To add a user to an enrollment, click **Add User**.
4. To delete a user from an enrollment, click **Delete** for that user.

FI Officer

The FI Officer receives tokens from the CBDC issuer and transfers them to other users.

An FI Officer user belongs to the `FI_CBDC_OFFICERS` group. The FI Officer receives tokens from the CBDC issuer and transfers them to users in any organization. The FI Officer also returns tokens to the CBDC Issuer. All transfers must be approved by the FI Manager. The FI Officer can navigate and see their own filterable transaction history.

The following tasks are typical ones that a FI Officer completes.

Transfer Tokens to CBDC Issuer

1. In the User Details pane, expand the central bank and then click **Deposit to <CentralBank>**. The Transfer <TokenID> CBDC to <CentralBank> window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

Transfer Tokens to Financial Institution Accounts

1. In the Transfer to FI Accounts pane, expand a financial institution, select a user, and then click **Transfer**. The Transfer <TokenID> CBDC to <FI Bank Name> window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

FI User

The FI User receives tokens from the officers of their bank.

An FI User user belongs to the `FI_CBDC_USERS` group. The FI User receives tokens from the FI Officer of their bank. FI Users can transfer tokens to users and officers of their own bank, and users at any other bank. Any transfers between banks must be approved by the FI Manager. The FI User can navigate and see their own filterable transaction history.

The following tasks are typical ones that a FI User completes.

Transfer Tokens

1. In the Transfer to FI Accounts pane, expand a financial institution, select a user, and then click **Transfer**. The Transfer <TokenID> CBDC to <FI Bank Name> window opens.
2. Enter a **Purpose** and a **CBDC Quantity**. All other fields are automatically populated and are read-only.
3. Click **Transfer <TokenID> Token**. The transaction runs and is displayed in the Transaction History pane.

FI Manager

The FI Manager approves or rejects hold requests for their bank.

An FI Manager user belongs to the `FI_CBDC_MANAGERS` group and has the `Notary` role. FI Managers approve or reject hold requests for their financial institution. Hold requests are used for transfers between banks.

Comprehensive details about transfer requests for their bank are displayed for the FI Manager. The Approve / Reject CBDC Transfers to Member Banks pane shows all pending transfers. The FI Manager can select some or all requests to approve or reject them. The Action History page displays all requests that have been approved or rejected by the FI Manager.

FI Auditor

The FI Auditor pplies auditing policies and reviews transactions that meet the specified policy conditions for their bank.

An FI Auditor user belongs to the `FI_CBDC_AUDITORS` group and has the `Org Auditor` role. The FI Auditor can navigate and see filterable transaction history for all accounts at their bank.

The following tasks are typical ones that a FI Auditor completes.

Apply Policies

1. In the Auditing Policies pane, select the policies to apply, as described in the following table.

Policy	Information
Single Transfer Amount	Select a symbol from the list and then enter an amount to audit single transfers.
Number of Transfers Exceeding	Enter a number of transfers and then select a period of time from the list to audit the number of transfers in that period.
Daily Transfer Total Amount Exceeding	Enter a value to audit the total daily transfer amount.
Any Transfers to Specific Account	Select a recipient to audit.
Any Transfers from Specific Accounts	Select a sender to audit.

2. Click **Apply**. The policies are applied and relevant transactions are displayed in the Transaction History pane.

4

Bond Marketplace Application

The bond marketplace application can support issuing and redemption as well as buying and selling of bonds, represented by fractional non-fungible tokens (NFTs).

The bond marketplace sample represents the life cycle of a traditional bond in the financial ecosystem. The sample specification file uses the extended ERC-1155 standard that is supported by Blockchain App Builder. The generated chaincode includes methods from initialization of a non-fungible bond token, account operations, and role assignments to bond issuance with a fixed coupon rate, bond purchases with wholesale CBDC, periodic coupon payment, and redemption of bonds at maturity with simple interest payments.

The bond marketplace solution comprises the following downloadable packages.

- Blockchain App Builder specification file
- Pre-built bond marketplace chaincode
- Wrapper APIs for bond setup and lifecycle operations

To get the bond marketplace packages, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Bond Marketplace**.

Bond Marketplace Chaincode Package

Oracle Blockchain Platform Digital Assets Edition includes sample chaincode for the bond NFT marketplace scenario.

The bond marketplace chaincode supports managing and trading bonds, represented by fractional non-fungible tokens (NFTs). You can use chaincode methods to issue, purchase, redeem, and trade bond NFTs in a decentralized environment.

The bond marketplace chaincode package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `BondMarketplace.zip`, an archive file that contains the packaged chaincode for deployment.
- `BondMarketplace.yaml`, a specification file that you can use with Blockchain App Builder to scaffold the `WholesaleCBDC` chaincode.
- `BondMarketplace_postman_collection.json`, a Postman collection that enables you to test the APIs in the chaincode.
- `README.md`, a step-by-step guide for working with the chaincode.

To get the bond marketplace chaincode package, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Bond Marketplace Application**.

Specification File

The bond marketplace specification file (`Bond_Marketplace.yml`) is based on the extended ERC-1155 specification file. It includes a `model` attribute, which generates the application-specific chaincode. In this case, `model: bond` creates additional methods for the bond marketplace application when the chaincode is generated. Additionally, specific parameters must be set in the metadata section of the file.

```
#
# Copyright (c) 2024, Oracle and/or its affiliates. All rights reserved.
#

# Token asset to manage the complete lifecycle of a Bond in a primary Bond
marketplace.

assets:
    - name: Bond #Asset name
      type: token #Asset type
      standard: erc1155+   # Token standard
      events: true # Supports event code generation for non-GET methods
      model: bond # Supports creation of additional methods for Primary Bond
marketplace

      anatomy:
          type: nonfungible # Token type
          unit: fractional  #Token unit

      behavior: # Bond token behaviors
        - divisible:
        - mintable:
        - transferable
        - burnable
        - roles:
            minter_role_name: minter
            burner_role_name: burner

      properties:  # Custom asset attributes for non-fungible Bond token.

          - name: status # Custom asset attribute maintains the status of the
Bond.
            type: string
            mandatory: true


      metadata: # To maintain the metadata on-chain, this tag will be used.
Users won't be able to update the metadata attribute values after the bond
token is minted.

          - name: ISIN # A unique alphanumeric code that identifies a
specific bond internationally.
            type: string
            mandatory: true

          - name: Segment # The classification of bonds based on issuer type
or purpose, such as corporate, government, sovereign, or green bonds.
            type: string

          - name: Issuer # The entity, such as a corporation or government,
that issues the bond.
            type: string
            mandatory: true
          - name: FaceValue # The principal amount of the bond that will be 
repaid at maturity.
            type: number
            mandatory: true

          - name: IssueSize # The total monetary value or quantity of bonds 
issued by the issuer.
            type: number
            mandatory: true

          - name: CouponRate # The annual interest rate that the bond pays, 
typically as a percentage of the face value.
            type: float
            mandatory: true

          - name: InterestPaymentType # Specifies whether the bond pays 
simple or compound interest.
            type: string
            mandatory: true
            validate: /^\\s*(simple)\\s*$/

          - name: InterestFrequency # The regularity with which interest 
payments are made, such as monthly, quarterly, annually or at maturity.
            type: string
            mandatory: true
            validate: /^\s*["]?((monthly|quarterly|annually|at maturity)\s*)
["]?\s*$/

          - name: IssueDate # The date when the bond was initially issued.
            type: date
            mandatory: true

          - name: MaturityDate # The date on which the bond’s principal 
amount will be repaid to the bondholder.
            type: date
            mandatory: true

customMethods:
```

Table 4-1   Metadata Parameters for the Bond Marketplace Specification File

Entry	Description
`name: ISIN`	A string that is a unique 12-character alphanumeric code that identifies a bond.
`name: Segment`	A string that represents the segment type of the bond.
`name: Issuer`	A string that represents the issuer of the bond.
`name: FaceValue`	A number that represents the face value (price) of the bond token.
`name: IssueSize`	A number that represents the issue size (total quantity) of the bond.

Table 4-1 (Cont.) Metadata Parameters for the Bond Marketplace Specification File

Entry	Description
`name: CouponRate`	A number that represents the coupon rate (interest rate) of the bond. It must be a per annum rate.
`name: InterestRateType`	A string that represents the interest payment type. The only supported value is `simple`.
`name: InterestFrequency`	A string that represents the interest frequency of the bond token. The following list shows the supported values. • `monthly` • `quarterly` • `annually` • `at maturity`
`name: IssueDate`	A date that represents the issue date of the bond.
`name: MaturityDate`	A date that represents the maturity date of the bond.

Deploy and Test Bond Marketplace Chaincode

Prerequisites

You must complete the following steps before you work with the bond marketplace chaincode.

1. Create a confidential client application in Oracle Identity Cloud Service. The bond marketplace wrapper API provides methods to create Identity Cloud Service users that can be used by the clients in their signup flow. This requires a confidential client application that has the privilege to create a new user in Identity Cloud Service. The bond marketplace chaincode requires the client ID and client secret of this application. For more information on adding a confidential application, see Add a Confidential Application.
2. Create user groups and configure them to the Oracle Blockchain Platform instance. Identity Cloud Service user groups such as `ADMIN_GROUP`, `USER_GROUP`, `CLIENT_GROUP`, and so on must be created and configured to the corresponding Oracle Blockchain Platform instances based on the corresponding access and application roles such as `admin`, `restproxy user`, `ca user`, and so on. For more information, see Set Up Users and Application Roles.
3. Deploy the wholesale CBDC chaincode. The wholesale CBDC chaincode must be deployed on the same Oracle Blockchain Platform instances where the bond marketplace chaincode is deployed. The corresponding token and token account must be set up in the wholesale CBDC chaincode as the default payment mode for the bond. You can do this manually or by using the Postman collection generated by Blockchain App Builder. The wholesale CBDC chaincode and the bond marketplace can be deployed on the same channel or on different channels.

Deploying the Chaincode

You can deploy the chaincode directly from the Oracle Blockchain Platform console or by using Blockchain App Builder. Before you deploy the chaincode, create enrollment IDs for each token user and then map the token users to their respective enrollment IDs. Specify only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to a REST Proxy.

When you deploy the token chaincode, you must call the `init` method and pass the organization ID and user ID of the `Token Admin` user.

For information about deploying from the Oracle Blockchain Platform console, see Use Advanced Deployment.

To deploy using Blockchain App Builder, complete the following steps.

1. Extract the `BondMarketplace.zip` archive file.
2. Import the `BondMarketplace` chaincode to the Blockchain App Builder extension in Visual Studio Code.
3. Edit the `.ochain.json` file to update the value of the `configFileLocation` key to the path of the `BondMarketplace.yml` specification file.
4. Open a terminal window and navigate to the chaincode folder, and then run the following command.

   ```
   npm install
   ```

For more information about deploying using Blockchain App Builder, see Deploy Your Chaincode Using Visual Studio Code.

Bond Marketplace Sample Process Flow

A typical process flow using the bond marketplace methods follows these basic steps.

1. Admins use the `createUserAccount` and `createTokenAccount` methods to create individual NFT accounts for all users.
2. Admins use the `addRole` method to assign the minter role to the officers of participating financial institutions (for example, `FI-1 Bond Issuer`).
3. Financial institution officers (bond issuers) use the `createBondToken` method to issue bonds as fractional NFTs.
4. Financial institution officers use the `getTokenById` method to review and confirm the details of issued bonds.
5. Bond purchasers (for example, `FI User`) use the `purchaseToken` method to buy the fractional NFT bond and to make a payment using the wholesale CBDC chaincode.
6. Purchasers use the `balanceOfBatch` method to verify the receipt of the bond in their wallet.
7. Purchasers use the `getAccountBalance` method (wholesale CBDC chaincode) to confirm the transfer in their wholesale CBDC wallet.
8. Bond issuers use the `payInterest` method to pay periodic interest to bondholders using the wholesale CBDC chaincode.
9. Purchasers use the `requestTokenRedemption` method to submit a request to redeem their bond after it has matured.
10. Financial institution officers use the `approveTokenRedemption` method to approve the redemption request and to transfer wholesale CBDC funds to the bond holder.
11. The bond holder uses the `balanceOfBatch` method (bond marketplace chaincode) and `getAccountBalance` method (wholesale CBDC chaincode) to verify that the bond was redeemed and that they received funds in their wholesale CBDC wallet.

For more details about using Postman collections, see the following topics.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Bond Marketplace Wrapper API Package

Oracle Blockchain Platform Digital Assets Edition includes a wrapper API package that extends the REST API to support operations specific to a bond NFT marketplace.

The wrapper API package uses the API Gateway service and OCI Functions to deploy API routes specifically for managing the bond marketplace NFT life cycle. The bond marketplace wrapper API package is downloadable from the Oracle Blockchain Platform console, and includes an archive file that contains the wrapper API package including the Terraform scripts required for deployment. You deploy this file to a Resource Manager stack on Oracle Cloud Infrastructure (OCI) to create the necessary Oracle resources for the wrapper APIs. It also includes a Postman collection that enables you to test the deployed wrapper APIs. The collection includes pre-configured requests with endpoints and payloads that correspond to the APIs defined in the wrapper API package.

Wrapper APIs

`activateAccount`
Original method name: `activateAccount`

This POST method activates a token account. This method can be called only by an admin or account owner. Deleted accounts cannot be activated.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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 JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
  "account_id":
```

```
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "active"
}
```

`addTokenAdmin`
Original method name: `addTokenAdmin`

This POST method adds other admins to the token chaincode. This method can be called only by the token admin of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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)"
}
```

`addRole`
Original method name: `addRole`

This POST method adds a role to a specified user and token. This method can be called only by a `Token Admin` of the chaincode. Non-fungible tokens are specified by the token name. The specified user must have a non-fungible token account. The specified role must exist in the specification file for the token.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "role": "role minter/burner"
    "tokenDetails": "{"tokenName": "token name value"}"
    "endorsers": {{endorsers}}
}
```

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 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~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a 
(Org-Id: appdev, User-Id: idcqa)"
}
```

`approveBondRedemption`
Original method name: `approveBondRedemption`

This POST method can be called only by the token creator to approve a request for the redemption of bond tokens. The approval operation transfers the bond NFT from the owner's account (the user who raised the request) to the creator's account, and transfers CBDC tokens from the bond creator's to the owner's account. Because of this, this method must be executed in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the account of the token owner who raised the redemption request, and the CBDC token transfer value must be equal to the calculated redemption price that was calculated by the chaincode while raising the redemption request.

Payload:

```
{
   "fromOrgId": "fromOrgId value",
   "fromUserId": "fromUserId value",
   "settlementId": "settlementId value",
   "tokenId": "{{bc-token-id}}",
   "CBDCTokenId": "CBDCTokenId value",
   "CBDCFromOrgId": "CBDCFromOrgId value",
   "CBDCFromUserId": "CBDCFromUserId value",
   "CBDCQuantity": 0,
   "CBDCRemark": "{\\\"category\\\":\\\"category value\\\",\\\"description\\
```

```
\":\\\"description value\\\"}",
   "endorsers": {{endorsers}}
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `CBDCTokenId: string` – The ID of the token in the CBDC chaincode.
- `CBDCOrgId: string` – The MSP ID of the user in the CBDC chaincode.
- `CBDCUserId: string` – The user name or email ID of the user in the CBDC chaincode.
- `CBDCQuantity: string` – The quantity of tokens to transfer in the CBDC chaincode.
- `CBDCRemark: string` – A remark for the transfer in the CBDC chaincode, which must be in the format shown previously.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{
```

```
"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

balanceOfBatch
Original method name: `balanceOfBatch`

This GET 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.

Query:

```
/balanceOfBatch?orgIds=["{{bc-org-id}}"]&userIds=["{{bc-user-
id}}"]&tokenIds=["{{bc-token-id}}"]
```

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~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "tokenAccountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "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
    }
]
```

`batchTransferFrom`
Original method name: `batchTransferFrom`

This POST method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user.

Payload:

```
{
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "toOrgId": "toOrgId value",
 "toUserId": "toUserId value",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "endorsers": {{endorsers}}
}
```

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)"
    }
]
```

`burnBatch`
Original method name: `burnBatch`

This POST method deactivates, or burns, the specified tokens. Any user with the burner role can call this method.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "sameOrgEndorser": true
}
```

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~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6 
(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)"
  }
]
```

`createAccount`
Original method name: `createAccount`

This POST method creates an account for a specified user and associated token accounts for 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "ftAccount": true,
 "nftAccount": true,
 "endorsers": {{endorsers}}
}
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `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"
}
```

`createAccountWithEnrollment`
Original method name: `createAccountWithEnrollment`

This POST method creates an enrollment for a user in the REST proxy of the instance, and creates an NFT account in the bond marketplace chaincode.

Payload:

```
{
 "orgId": "orgId value",
 "userId": "userId value"
}
```

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 that was created.

Return Value Example:

```
{
 "blockNumber": 44,
 "encode": "JSON",
 "payload": {
 "accountId":
"ouaccount~1930ec223036c0fe2ea97c58fd9a8d2456d13c0fd0c98217ce075ceddb8add02",
 "assetType": "ouaccount",
 "associatedFtAccounts": [],
 "associatedNftAccount":
"oaccount~03f8a6949f6c5c453354a4a8eed8503a39766085b476430e95ce305769fba861",
 "orgId": "BondMPTest",
 "totalAccounts": 1,
 "totalFtAccounts": 0,
 "userId": "u14"
 },
 "sourceURL": "bondmptest-oabcs1-iad.blockchain.ocp.example.com:20010",
 "txid": "9fa75a631f0de2ddf2ed85742dcc4d4b97b7afb0d3f9a07988e5cb9120ed633f"
}
```

`createBondToken`
Original method name: `createBondToken`

This POST method creates tokens. Every token that is defined has its own create method. For non-fungible tokens, the caller of this method becomes the owner of the NFT. If `roles` behavior is defined in the token model `behaviors` property and a `minter_role_name` is specified, the caller account must have the minter role.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"tokenUri\":\"tokenUri value\",\"tokenMetadata\":{\"ISIN\":\"ISIN
value\",\"Segment\":\"Segment value\",\"Issuer\":\"Issuer
value\",\"FaceValue\":999,\"IssueSize\":999,\"CouponRate\":999,\"InterestPayme
ntType\":\"InterestPaymentType
value\",\"InterestFrequency\":\"InterestFrequency
value\",\"IssueDate\":\"2023-03-28T15:16:36+00:00\",\"MaturityDate\":\"2023-03
-28T15:16:36+00:00\"},\"status\":\"status value\"}",
 "quantity": 1,
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.
- `quantity: number` – The number of tokens to mint. The only supported value for this parameter is `1`.

Returns:

- On success, a JSON object of the account that was created.

Return Value Example:

```
{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
```

```
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

`createIDSCUser`
Original method name: `createIDCSUser`

This POST method creates an Identity Cloud Service user in the tenancy specified by the URL in the `terraform.tfvars` file and assigns the user to the specified user group.

Payload:

```
{
 "userName": "userName value",
 "firstName": "firstName value",
 "lastName": "lastName value",
 "email": "email value",
 "groupName": "groupName value"
}
```

Parameters:

- `userName: string` – The ID of the user.
- `firstName: string` – The first name of the user.
- `lastName: string` – The last name of the user.
- `email: string` – The email address of the user.
- `groupName: string` – The name of the Identity Cloud Service group to assign to the user.

Return Value Example:

```
{
 "status": "Success",
 "msg": "User user1 is created and assigned to the group BOND_ADMIN"
}
```

`createTokenAccount`
Original method name: `createTokenAccount`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenType": "nonfungible",
 "endorsers": {{endorsers}}
}
```

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~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "tokenId": ""
    }
  ],
  "associatedNftAccount": ""
}
```

`createUserAccount`
Original method name: `createUserAccount`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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": ""
}
```

`deleteAccount`
Original method name: `deleteAccount`

This POST method deletes a token account. This method can be called only by a `Token Admin` of the chaincode. This method throws an error if an `accountStatus` value for the account is not found in the ledger.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
  "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

`getAccount`
Original method name: `getAccount`

This GET 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.

Query:

```
/getAccount?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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.

Return Value Example

```
{
    "assetType": "oaccount",
    "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
    "userId": "user2",
    "orgId": "AppBldFFFFMay22",
    "tokenType": "nonfungible",
    "noOfNfts": 3
}
```

`getAccountBondSummary`
Original method name: `getAccountBondSummary`

This GET method returns an account summary for the specified user, including details of purchased or redeemed tokens and their purchase and redemption prices.

Query:

```
/getAccount?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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 that includes the token account summary.

Return Value Example

```
[
 {

"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"REJECTED"
      },
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "tokenId":"bond1",
         "status":"Purchased",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op2",
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"APPROVED"
      }
   ]
```

getAccountBondSummaryWithPagination
Original method name: `getAccountBondSummaryWithPagination`

This GET method returns an account summary for the specified user, including details of purchased or redeemed tokens and their purchase and redemption prices. This method can return results with pagination based on pagesize and bookmark values, and also filtered by start time and end time.

Query:

```
/getAccountBondSummary?orgId={{bc-org-id}}&userId={{bc-user-
id}}&pageSize=1&bookmark={{bookmark}}
```

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.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Returns:

- On success, a JSON object that includes the token account summary.

Return Value Example

```
[
 {

"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"REJECTED"
      },
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "tokenId":"bond1",
         "status":"Purchased",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op2",
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"APPROVED"
      }
   ]
```

getAccountDetailsByUser
Original method name: `getAccountDetailsByUser`

This GET 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.

Query:

```
/getAccountDetailsByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
    "associatedFTAccounts": [
        {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "tokenId": "FT",
            "balance": 50
        }
    ],
    "associatedNFTAccount": {
        "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "associatedNFTs": [
            {
                "nftTokenId": "FNFT",
                "tokenShare": 100
            },
            {
                "nftTokenId": "FNFT2",
                "tokenShare": 110
            },
            {
                "nftTokenId": "NFT"
            }
        ]
    }
}
```

`getAccountStatus`
Original method name: `getAccountStatus`

This GET method retrieves 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.

Query:

```
/getAccountStatus?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
    "accountId": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

`getAccountStatusHistory`
Original method name: `getAccountStatusHistory`

This GET method retrieves the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

Query:

```
/getAccountStatusHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
      "accountId": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "suspended"
    }
  },
  {
    "trxId": 
```

```
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
      "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "active"
    }
  }
]
```

`getAccountTransactionHistory`
Original method name: `getAccountTransactionHistory`

This GET method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
/getAccountTransactionHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddfae140bc
~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475b",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 10,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "transactionType": "DEBIT",
        "balance": 90
    },
    {
        "transactionId":
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddfae140bc
~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc114",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "transactionType": "DEBIT"
    },
    {
        "transactionId":
"otransaction~c369929e28e78de06c72d020f1418c9a154a7dd280b2e22ebb4ea4485e249124
~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "timestamp": "2023-06-06T14:47:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT"
    },
    {
        "transactionId":
"otransaction~114a1bc78d04be48ee6dc140c32c042ee9481cb118959626f090eec744522422
~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "timestamp": "2023-06-05T17:17:57.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 100,
        "triggeredByUserAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT",
        "balance": 100
    }
]
```

`getAccountsByRole`
Original method name: `getAccountsByRole`

This GET method returns a list of all account IDs for a specified role and token.

Query:

```
/getAccountsByRole?role=role value (for example minter /
burner)&tokenDetail={"tokenName":"tokenName value"}
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetail: JSON` – For fungible tokens, the token ID. For non-fungible tokens, the require token name.

Return Value Example:

```
{
  "accounts": [
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",

"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b"
  ]
}
```

`getAllAccounts`
Original method name: `getAllAccounts`

This GET method returns details of all user accounts. This method can be called only by a `Token Admin` of the chaincode.

Query:

```
/getAllAccounts
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
        {
            "assetType": "ouaccount",
            "accountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
            "userId": "user2",
            "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"
        },
    ]
```

`getAllTokenAdmins`
Original method name: `getAllTokenAdmins`

This GET 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.

Query:

```
/getAllTokenAdmins
```

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"
    }
  ]
}
```

`getAllTokens`
Original method name: `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.

```
/getAllTokens
```

Parameters:

- none

Returns:

- A list of all token assets in JSON format.

Return Value Example:

```
[{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "status value"
}]
```

getAllTokensByUser
Original method name: `getAllTokensByUser`

This GET 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.

Query:

```
/getAllTokensByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
[{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "status value"
}]
```

`getAllTokensWithFilters`
Original method name: `getAllTokensWithFilters`

The admin can call this GET method to fetch all the tokens filtered by status.

Query:

```
/getAllTokensWithFilters?status=status&pageSize=pageSize&bookmark=bookmark
```

Parameters:

- `status: string` – The status of the token, which can either be `CREATED` or `POSTED`.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }

]
```

`getTokenApprovalRequestByUser`
Original method name: `getTokenApprovalRequestByUser`

Any account holder can call this GET method to get the details of all the token approval requests (redemption requests) they have made.

Query:

```
/getTokenApprovalRequestByUser?status=status value
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

`getTokenApprovalRequestForUserByStatus`
Original method name: `getTokenApprovalRequestByUser`

Any account holder can call this GET method to get the details of all the token approval requests (redemption requests) they have made.

Query:

```
/getTokenApprovalRequestForUserByStatus?status=status value
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

`getTokenById`
Original method name: `getTokenById`

This GET 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.

Query:

```
/getTokenById?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }

]
```

`getTokenHistory`
Original method name: `getTokenHistory`

This GET method returns the history for a specified token ID.

Query:

```
/getTokenHistory?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- On success, a JSON array that contains the token history.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }

]
```

getUsersByRole
Original method name: `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.

Query:

```
/getUsersByRole?role=role value (for example minter /
burner)&tokenDetail={"tokenName":"tokenName value"}
```

Parameters:

- `role: string` – The name of the role to search for.

- `tokenDetail: JSON` – For fungible tokens, the token ID. For non-fungible tokens, the require token name.

Return Value Example:

```
{
    "users": [
        {
            "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
            "orgId": "appdev",
            "userId": "user2"
        },
        {
            "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
            "orgId": "appdev",
            "userId": "user1"
        }
    ]
}
```

`init`
Original method name: `init`

This POST 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.

Payload:

```
{
 "adminList": "[{\"orgId\":\"{{bc-org-id}}\",\"userId\":\"{{bc-user-id}}\"}]"
}
```

Parameters:

- `adminList array` – An array of `{orgId, userId}` information that specifies the list of token admins. The `adminList` array is a mandatory parameter.

Returns:

- On success, a message with no payload.

Return Value Example:

```
{
}
```

isInRole
Original method name: `isInRole`

This GET method returns a Boolean value to indicate if a user has a specified role. 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.

Query:

```
/isInRole?orgId={{bc-org-id}}&userId={{bc-user-id}}&role=role value (for
example minter / burner)&tokenDetail={"tokenName":"tokenName value"}
```

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 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"
}
```

isTokenAdmin
Original method name: `isTokenAdmin`

This GET 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.

```
/isTokenAdmin?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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}
```

`mintBatch`
Original method name: `mintBatch`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "sameOrgEndorser": true
}
```

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~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"
        }
    ]
}
```

`ownerOf`
Original method name: `ownerOf`

This GET method returns the account ID, organization ID, and user ID of the owner of the specified token ID. Anyone can call this method.

Query:

```
/ownerOf?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
[
    {
        "accountId": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "orgId": "Org1MSP",
        "userId": "admin"
    },
    {
        "accountId": 
"oaccount~74108eca702bab6d8548e740254f2cc7955d886885251d52d065042172a59db0",
        "orgId": "Org1MSP",
        "userId": "user"
    }
]
```

`payInterest`
Original method name: `ownerOf`

This POST method can be called only by the token creator or admin to pay the interest earned on the bond token. This method can be called only if the interest frequency of the token is monthly, quarterly, or annually. Interest cannot be paid if the interest frequency is at maturity. Interest is calculated by the chaincode based on the coupon rate of the token. The purchase operation transfers CBDC tokens from the caller's account to the bond owner's account. Because of this, this method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token owner, and the CBDC token transfer value must be equal to the interest calculated by the bond chaincode.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenId": "{{bc-token-id}}",
 "orderId": "orderId value",
 "CBDCTokenId": "CBDCTokenId value",
 "CBDCFromOrgId": "CBDCFromOrgId value",
 "CBDCFromUserId": "CBDCFromUserId value",
 "CBDCQuantity": 0,
 "CBDCRemark": "{\\\"category\\\":\\\"category value\\\",\\\"description\\\":\
\\"description value\\\"}",
 "endorsers": {{endorsers}}
}
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the operation.
- `CBDCTokenId: string` – The ID of the token in the CBDC chaincode.
- `CBDCOrgId: string` – The MSP ID of the user in the CBDC chaincode.
- `CBDCUserId: string` – The user name or email ID of the user in the CBDC chaincode.
- `CBDCQuantity: string` – The quantity of tokens to transfer in the CBDC chaincode.
- `CBDCRemark: string` – A remark for the transfer in the CBDC chaincode, which must be in the format shown previously.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

`postBondToken`
Original method name: `postBondToken`

This POST method can be called only by a token creator. The method submits the bond token for listing in the marketplace. When a token is created, its status is initially set to `created`. This method updates the status to `posted`. Users can run the `getAllTokensWithFilter` method to retrieve all NFTs with a `posted` status.

Payload:

```
{
 "tokenId": "{{bc-token-id}}",
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to post.

Return Value Example:

```
{
         "isValid":true,
         "payload":{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"created"
         },
         "message":"Successfully updated asset with ID bond1"
      }
```

`purchaseBondToken`
Original method name: `purchaseBondToken`

This POST method can be called by any account holder to purchase a listed bond NFT. The purchase transfers the bond NFT from the creator's account to the caller's account, and transfers CBDC tokens from the caller's account to the creator's account. Because of this, the method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring that the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token creator, and the CBDC token transfer value must be equal to the face value of the bond token multiplied by the quantity being purchased.

Payload:

```
{
 "tokenId": "{{bc-token-id}}",
 "quantity": 1,
 "orderId": "orderId value",
 "additionalFees": 1,
 "CBDCTokenId": "CBDCTokenId value",
 "CBDCFromOrgId": "CBDCFromOrgId value",
 "CBDCFromUserId": "CBDCFromUserId value",
 "CBDCQuantity": 0,
 "CBDCRemark": "{\\\"category\\\":\\\"category value\\\",\\\"description\\\":\
\\"description value\\\"}",
 "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token to purchase.
- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the operation.
- `CBDCTokenId: string` – The ID of the token in the CBDC chaincode.
- `CBDCOrgId: string` – The MSP ID of the user in the CBDC chaincode.
- `CBDCUserId: string` – The user name or email ID of the user in the CBDC chaincode.
- `CBDCQuantity: string` – The quantity of tokens to transfer in the CBDC chaincode.
- `CBDCRemark: string` – A remark for the transfer in the CBDC chaincode, which must be in the format shown previously.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

`rejectBondRedemption`
Original method name: `rejectBondRedemption`

The token creator can call this POST method to reject the redemption request. Token owners can raise a redemption request again by using a different settlement ID.

Payload:

```
{
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "settlementId": "settlementId value",
 "tokenId": "{{bc-token-id}}",
 "endorsers": {{endorsers}}
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully rejected the token approval request"
}
```

`removeRole`
Original method name: `removeRole`

This POST 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.

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "role": "role value (for example minter / burner)",
 "tokenDetail": "{\"tokenName\":\"tokenName value\"}",
 "endorsers": {{endorsers}}
}
```

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 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)"
}
```

`removeTokenAdmin`
Original method name: `removeTokenAdmin`

This POST 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`.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "sameOrgEndorser": true
}
```

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)"}
```

`requestBondRedemption`
Original method name: `requestBondRedemption`

This POST method can be called only by the token owner to raise a request for the redemption of bond tokens after maturity. This method is also involved the calculation of the redemption price by the chaincode. Redemption requests can be raised only on the entire quantity of the bond token that the user owns. Users can raise multiple redemption requests based on different settlement IDs but only one can be approved by the token creator.

Payload:

```
{
 "settlementId": "settlementId value",
 "tokenId": "{{bc-token-id}}",
 "orderId": "orderId value",
 "additionalFees": 1,
 "endorsers": {{endorsers}}
}
```

Parameters:

- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the redemption price.

Return Value Example:

```
{
         "tokenName":"bond",
         "assetType":"otokenApproval",

"id":"otokenApproval~9e006057ac96ae997e3964531b1a08ad2316555701c7fe9ec7b88e38e
20892bf",
         "settlementId":"op4",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op4",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
         "tokenId":"bond1",
         "quantity":1,
         "status":"PENDING",
         "orderId":"op4",
         "redeemPrice":11

}
```

`safeBatchTransferFrom`
Original method name: `safeBatchTransferFrom`

This POST 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.

```
{
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "toOrgId": "toOrgId value",
 "toUserId": "toUserId value",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "endorsers": {{endorsers}}
}
```

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)"
    }
]
```

suspendAccount
Original method name: `suspendAccount`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
    "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "suspended"
}
```

updateBondToken
Original method name: `updateBondToken`

This POST 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. This method can be called only by the token owner.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"tokenUri\":\"tokenUri value\",\"status\":\"status value\",
\"tokenMetadata\":{\"ISIN\":\"ISIN value\",\"Segment\":\"Segment
value\",\"Issuer\":\"Issuer
value\",\"FaceValue\":999,\"IssueSize\":999,\"CouponRate\":999,\"InterestPayme
ntType\":\"InterestPaymentType
value\",\"InterestFrequency\":\"InterestFrequency
value\",\"IssueDate\":\"2023-03-28T15:16:36+00:00\",\"MaturityDate\":\"2023-03
-28T15:16:36+00:00\"},\"status\":\"status value\"}",
 "sameOrgEndorser": true
}
```

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": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

URI
Original method name: `URI`

This method returns the URI of a specified token.

Query:

```
/URI?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "tokenUri": "example.com"
}
```

The following table shows the mapping between the chaincode API names and the updated endpoints. For detailed information about the chaincode API, see Scaffolded TypeScript Token Project.

Updated endpoint	TypeScript chaincode API name	Go chaincode API name	Method type
`activateAccount`	`activateAccount`	`ActivateAccount`	Native ERC-1155 method
`addAdmin`	`addAdmin`	`AddAdmin`	Native ERC-1155 method
`addRole`	`addRole`	`AddRole`	Native ERC-1155 method
`approveBondRedemption`	`approveTokenRedemption`	`ApproveTokenRedemption`	Modified
`balanceOfBatch`	`balanceOfBatch`	`BalanceOfBatch`	Native ERC-1155 method
`batchTransferFrom`	`batchTransferFrom`	`BatchTransferFrom`	Native ERC-1155 method
`burnBatch`	`burnBatch`	`BurnBatch`	Native ERC-1155 method
`createAccount`	`createAccount`	`CreateAccount`	Native ERC-1155 method
`createAccountWithEnrollment`	`createAccountWithEnrollment`	`CreateAccountWithEnrollment`	Added

Updated endpoint	TypeScript chaincode API name	Go chaincode API name	Method type
`createBondToken`	`createBondToken`	`CreateBondToken`	Native ERC-1155 method
`createIDCSUser`	`createIDCSUser`	`CreateIDCSUser`	Added
`createTokenAccount`	`createTokenAccount`	`CreateTokenAccount`	Native ERC-1155 method
`createUserAccount`	`createUserAccount`	`CreateUserAccount`	Native ERC-1155 method
`deleteAccount`	`deleteAccount`	`DeleteAccount`	Native ERC-1155 method
`getAccount`	`getAccount`	`GetAccount`	Native ERC-1155 method
`getAccountBondSummary`	`getAccountBondSummary`	`GetAccountBondSummary`	Bond marketplace method
`getAccountBondSummaryWithPagination`	`getAccountBondSummaryWithPagination`	`GetAccountBondSummaryWithPagination`	Bond marketplace method
`getAccountDetailsByUser`	`getAccountDetailsByUser`	`GetAccountDetailsByUser`	Native ERC-1155 method
`getAccountStatus`	`getAccountStatus`	`GetAccountStatus`	Native ERC-1155 method
`getAccountStatusHistory`	`getAccountStatusHistory`	`GetAccountStatusHistory`	Native ERC-1155 method
`getAccountTransactionHistory`	`getAccountTransactionHistory`	`GetAccountTransactionHistory`	Native ERC-1155 method
`getAccountsByRole`	`getAccountsByRole`	`GetAccountsByRole`	Native ERC-1155 method
`getAllAccounts`	`getAllAccounts`	`GetAllAccounts`	Native ERC-1155 method
`getAllTokenAdmins`	`getAllTokenAdmins`	`GetAllTokenAdmins`	Native ERC-1155 method
`getAllTokens`	`getAllTokens`	`GetAllTokens`	Native ERC-1155 method
`getAllTokensByUser`	`getAllTokensByUser`	`GetAllTokensByUser`	Native ERC-1155 method
`getAllTokensWithFilters`	`getAllTokensWithFilters`	`GetAllTokensWithFilters`	Bond marketplace method
`getTokenApprovalRequestByUser`	`getTokenApprovalRequestByUser`	`GetTokenApprovalRequestByUser`	Bond marketplace method
`getTokenApprovalRequestForUserByStatus`	`getTokenApprovalRequestForUserByStatus`	`GetTokenApprovalRequestForUserByStatus`	Bond marketplace method
`getTokenById`	`getTokenById`	`GetTokenById`	Native ERC-1155 method
`getTokenHistory`	`getTokenHistory`	`GetTokenHistory`	Native ERC-1155 method
`getUsersByRole`	`getUsersByRole`	`GetUsersByRole`	Native ERC-1155 method
`init`	`init`	`Init`	Native ERC-1155 method
`isInRole`	`isInRole`	`IsInRole`	Native ERC-1155 method

Updated endpoint	TypeScript chaincode API name	Go chaincode API name	Method type
`isTokenAdmin`	`isTokenAdmin`	`IsTokenAdmin`	Native ERC-1155 method
`mintBatch`	`mintBatch`	`MintBatch`	Native ERC-1155 method
`ownerOf`	`ownerOf`	`OwnerOf`	Native ERC-1155 method
`payInterest`	`payInterest`	`PayInterest`	Modified
`postBondToken`	`postToken`	`PostToken`	Bond marketplace method
`purchaseBondToken`	`purchaseToken`	`PurchaseToken`	Modified
`rejectBondRedemption`	`rejectTokenRedemption`	`RejectTokenRedemption`	Bond marketplace method
`removeRole`	`removeRole`	`RemoveRole`	Native ERC-1155 method
`removeTokenAdmin`	`removeTokenAdmin`	`RemoveTokenAdmin`	Native ERC-1155 method
`requestBondRedemption`	`requestTokenRedemption`	`RequestTokenRedemption`	Bond marketplace method
`safeBatchTransferFrom`	`safeBatchTransferFrom`	`SafeBatchTransferFrom`	Native ERC-1155 method
`suspendAccount`	`suspendAccount`	`SuspendAccount`	Native ERC-1155 method
`updateBondToken`	`updateBondToken`	`UpdateBondToken`	Native ERC-1155 method
`URI`	`URI`	`URI`	Native ERC-1155 method

Customize Wrapper APIs for Bond Marketplace

The bond marketplace wrapper API is a modified version of the wrapper API package that is generated by Blockchain App Builder.

Complete the following steps if you regenerate the wrapper API package after adding custom methods to the bond marketplace chaincode. To ensure that your changes are compatible, you must modify the newly generated wrapper API package by using the bond marketplace wrapper API package that is bundled with the product.

1. Use Blockchain App Builder to generate a wrapper API package for the bond marketplace chaincode.
2. Extract the files from the package.
3. Copy the following folders from the wrapper API package that is bundled with the product into the directory structure of the newly generated wrapper API package.
   - `createIDCSUser`
   - `createAccountWithEnrollment`
   - `purchaseBondToken` or `purchaseToken`
   - `` `approveBondRedemption `` or `` `approveedemption ``
   - `payInterest`

4. Add the following variables to the terraform.vars file.

   ```
   idcs_url="<idcs_url>"
    idcs_port="<idcs_port>"
    cbdc_chaincode="<cbdc_chaincode>"
    cbdc_channel="<cbdc_channel>"
    prepare_timeout=<prepare_timeout>
    isolation_level="<isolation_level>"
   ```

5. Add an entry at the end of the `terraform.vars` file for the `createIDCSUser` and `createAccountWithEnrollment` methods, as shown in the following example JSON string.

   ```
   \"createIDCSUser\":{\"path\":\"/createIDCSUser\",\"type\":[\"POST\"]},
        \"createAccountWithEnrollment\":{\"path\":\"/
   createAccountWithEnrollment\",\"type\":[\"POST\"]}
   ```

   The following text shows the general format of the JSON string in the *function_path* variable in the `terraform.vars` file.

   ```
   {"<methodName>":{"path":"/<methodFolderName>","type":["<HTTP Method POST
   or GET>"]}}
   ```

6. Verify and update the names for the modified methods `purchaseBondToken`, `approveBondRedemption`, `payInterest` to ensure that they are named correctly in the *function_path* variable in the `terraform.vars` file.
7. Replace the `main.tf` file in the newly generated wrapper API package with the `main.tf` file from the wrapper API package that is bundled with the product.

Deploy and Test Wrapper APIs for Bond Marketplace

Deploying the Wrapper API Package

Before you can deploy the wrapper API package, you must update the required configuration variables. Some configuration variables have default values, but you must manually update any variable that contains a placeholder as its default value. Configuration variables are stored in the `terraform.tfvars` file in the wrapper API archive. For more information about deploying wrapper APIs and about configuration variables, see Wrapper APIs and Wholesale CBDC Wrapper API Package. The following table lists the configuration variables and their defaults values for the bond marketplace wrapper API package.

Variable name	Default value	Description
`compartment_ocid`	`<compartment_ocid>`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	`<compartment_name>`	The name of the OCI compartment.
`identity_domain`	`<identity_domain>`	The identity domain to use.
`blockchain_channel`	`<blockchain_channel>`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	`<blockchain_url>`	The Oracle Blockchain Platform URL associated with the chaincode deployment.

Variable name	Default value	Description
`idcs_url`	`<idcs_url>`	The URL of the Identity Cloud Service.
`idcs_port`	`<idcs_port>`	The port number of the Identity Cloud Service.
`cbdc_chaincode`	`<cbdc_chaincode>`	The name of the wholesale CBDC chaincode.
`cbdc_channel`	`<cbdc_channel>`	The channel where the wholesale CBDC chaincode is deployed.
`isolation_level`	`<isolation_level>`	The isolation level for atomic transactions. Typically, this is `serializable`..
`blockchain_chaincode`	`WholesaleCBDC`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	`true`	The sync value to include in the payload for API calls.
`blockchain_timeout`	`6000`	The timeout value to include in the payload for API calls.
`vcn_display_name`	`WholesaleCBDC`	The display name of the OCI virtual cloud network.
`application_display_name`	`WholesaleCBDC`	The display name of the OCI application.
`gateway_display_name`	`WholesaleCBDC`	The display name of API Gateway.
`deployment_display_name`	`WholesaleCBDC`	The display name of the deployment in API Gateway.
`deployment_path_prefix`	`/WholesaleCBDC`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	`wholesalecbdc`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.
`policy_name`	`WholesaleCBDC`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

Bond Marketplace Sample Process Flow

A typical process flow using the bond marketplace wrapper APIs follows these basic steps.

1. Admins use the `createUserAccount` and `createTokenAccount` methods to create individual NFT accounts for all users.
2. Admins use the `addRole` method to assign the minter role to the officers of participating financial institutions (for example, `FI-1 Bond Issuer`).
3. Financial institution officers (bond issuers) use the `createBondToken` method to issue bonds as fractional NFTs.

4. Financial institution officers use the `getTokenById` method to review and confirm the details of issued bonds.
5. Bond purchasers (for example, `FI User`) use the `atomicTransaction` method to call the `purchaseToken` method to buy the fractional NFT bond and to make a payment using the wholesale CBDC chaincode.
6. Purchasers use the `balanceOfBatch` method to verify the receipt of the bond in their wallet.
7. Purchasers use the `getAccountBalance` method (wholesale CBDC chaincode) to confirm the transfer in their wholesale CBDC wallet.
8. Bond issuers use the `atomicTransaction` method to call the `payInterest` method to pay periodic interest to bondholders using the wholesale CBDC chaincode.
9. Purchasers use the `requestTokenRedemption` method to submit a request to redeem their bond after it has matured.
10. Financial institution officers use the `atomicTransaction` method to call the `approveTokenRedemption` method to approve the redemption request and to transfer wholesale CBDC funds to the bond holder.
11. The bond holder uses the `balanceOfBatch` method (bond marketplace chaincode) and `getAccountBalance` method (wholesale CBDC chaincode) to verify that the bond was redeemed and that they received funds in their wholesale CBDC wallet.

Postman Collection

The Postman collection in the bond marketplace wrapper API package includes additional attributes and methods that support the bond marketplace chaincode. For more information, see Wrapper API Package Components.

5

Generic Token Frameworks

Oracle Blockchain Platform Digital Assets Edition includes chaincode and wrapper APIs for generic token applications.

Fungible Token Framework

The fungible token framework uses the extended Token Taxonomy Framework standard that is supported by Blockchain App Builder.

Fungible tokens are digital assets that are interchangeable and uniform in value. Blockchain App Builder extends ERC-20 specification, which is based on the Token Taxonomy Framework (TTF), to support all phases of the fungible token life cycle including creation, development, management, and maintenance.

Example use cases:

Digital Currencies
Issuing digital representations of fiat currencies such as central bank digital currency (CBDC), stablecoins, or deposit tokens for efficient transactions in a blockchain network.

Loyalty Programs
Creating reward points that can be earned and redeemed by customers, enhancing customer engagement.

Micro-Payments
Enabling small-value transactions for blockchain transactions, marketplace services, Internet of Things (IoT) systems, or content monetization platforms.

The enhanced version of Blockchain App Builder that is included with Oracle Blockchain Platform Digital Assets Edition supports the following functions.

- Fractional units: Divide tokens into smaller units, allowing for precise value representation.
- Multiple fungible tokens: Create multiple fungible token types in a single smart contract (for example, multiple currencies or rewards points).
- Minting and burning: Control the token supply in circulation by creating (minting) and removing (burning) tokens.
- Account management: Manage account statuses, including activation, suspension, and deletion, to ensure compliance and security. One user can have multiple token accounts based on the various token types.
- Compliance controls: Enforce account-level daily limits and run auditing procedures to adhere to regulatory requirements.
- Transfer: Move a specified amount of fungible tokens between accounts.
- Notary accounts: Require an additional approval step during minting, burning and transfer operations, adding an extra layer of authorization to implement the maker-checker principle.

- Exchange pools: Exchange different types of tokens or assets atomically using liquidity pools in the blockchain network.
- Role operations: Assign and enforce roles such as minter, burner, notary, auditor, and organization auditor to provide specific privileges to any user account.

Oracle Blockchain Platform Digital Assets Edition includes a chaincode package and a wrapper API package for the fungible token scenario. The chaincode package includes a deposit token sample, which illustrates use of the framework. The wrapper API package extends the REST API to support operations specific to the deposit token scenario.

Fungible Token Framework Chaincode Package

The fungible token framework uses the extended Token Taxonomy Framework standard that is supported by Blockchain App Builder.

The deposit token sample illustrates the use of the fungible token generic framework, which is based on the extended Token Taxonomy Framework standard supported by Blockchain App Builder. The sample represents a system where deposit tokens represent fiat currency held at financial institutions and are issued and managed by regulated financial institutions. The sample specification file generates methods for initializing a deposit token, managing accounts, assigning roles, and performing operations such as minting, transferring, and burning tokens. It also provides notary accounts for approving minting and transfers, enforces compliance with account-level daily limits, and enables auditing procedures.

The fungible token framework chaincode package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `DepositToken.zip`, an archive file that contains the packaged chaincode for deployment.
- `DepositToken.yaml`, a specification file that you can use with Blockchain App Builder to scaffold the `DepositToken` chaincode.
- `DepositToken_postman_collection.json`, a Postman collection that enables you to test the APIs in the chaincode.
- `README.md`, a step-by-step guide for working with the chaincode.

To get the fungible token framework, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Fungible Token Framework**.

For more details about using Postman collections, see the following topics.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Specification File

The specification file that is used to generate the deposit token chaincode includes the `events` attribute. The chaincode events function supports event callbacks in generated chaincodes to enable real-time notifications and trigger workflows. For more information about specification files and the parameters used in specification files, see Input Specification File for Fungible Tokens in *Using Oracle Blockchain Platform*.

The deposit token chaincode is based on the extended Token Taxonomy Framework standard, with customizations to support the application scenario. The following behavior section of the specification file is required to enable these customizations.

```
behavior: # Token behaviors
    - divisible:
        decimal: 2
    - mintable:
        mint_approval_required: true
    - transferable
    - burnable
    - holdable
    - roles:
        minter_role_name: minter
        notary_role_name: notary
        mint_approver_role_name: notary
```

The following code is the specification file for the deposit tokens sample.

```
#
# Copyright (c) 2024, Oracle and/or its affiliates. All rights reserved.
#

assets:

# This specification file is an example how to build any fungible token
application.
# For a fungible token application, deposit token system has been used as an
example.
# Deposit token is a digital representation of deposits held at commercial
banks, enabling transactions on blockchain networks while maintaining the
value and stability of traditional bank deposits.


    - name: Deposit # Asset name
      type: token  # Asset type
      standard: ttf+   # Token standard
      events: true # Supports event code generation for non-GET methods

      anatomy:
          type: fungible # Token type
          unit: fractional # Token unit

      behavior: # Token behaviors
          - divisible:
              decimal: 2
          - mintable:
              mint_approval_required: true
          - transferable
          - holdable
          - burnable
          - roles:
              minter_role_name: minter
              notary_role_name: notary
              mint_approver_role_name: notary
      properties:
          - name: Token_Name # Custom attribute to represent the deposit
token name.
            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

customMethods:
```

Endorser Details in Chaincode Methods

Oracle Blockchain Platform Digital Assets Edition adds an endorsement parameter to the request payload for all setter methods. The value of the parameter is either `endorsers` or `sameOrgEndorser`. If the `sameOrgEndorser` parameter is true, transaction endorsements must be from the same organization as the requester. The `endorsers` parameter specifies a list of peers that must endorse the transaction. For more information, see Endorsement Support in Postman Collections. The following table shows the endorser type for each method.

Method	Endorser Type
`activateAccount`	`endorsers`
`addTokenAdmin`	`sameOrgEndorser`
`addTokenAuditor`	`sameOrgEndorser`
`addOrgAdmin`	`sameOrgEndorser`
`addOrgAuditor`	`sameOrgEndorser`
`addRole`	`endorsers`
`approveMint`	`sameOrgEndorser`
`executeHoldTokens`	`endorsers`
`associateTokenToAccount`	`endorsers`
`createAccount`	`endorsers`
`getAccount`	`endorsers`
`getAccountBalance`	`endorsers`
`getAccountsByUser`	`endorsers`
`getAccountTransactionHistory`	`endorsers`
`getAccountTransactionHistoryWithFiltersFromRichHistDB`	`endorsers`
`getNetTokens`	`endorsers`
`getOnHoldIds`	`endorsers`
`getTotalMintedTokens`	`endorsers`
`getUserByAccountId`	`endorsers`
`getUsersByRole`	`endorsers`
`holdTokens`	`endorsers`
`init`	`endorsers`
`initializeDepositToken`	`sameOrgEndorser`
`issueTokens`	`sameOrgEndorser`

Method	Endorser Type
rejectMint	sameOrgEndorser
releaseHoldTokens	endorsers
removeTokenAdmin	sameOrgEndorser
removeTokenAuditor	sameOrgEndorser
removeOrgAdmin	sameOrgEndorser
removeOrgAuditor	sameOrgEndorser
removeRole	endorsers
requestMint	sameOrgEndorser
burnTokens	sameOrgEndorser
setMaxDailyAmount	endorsers
setMaxDailyTransactionCount	endorsers
suspendAccount	endorsers
transferTokens	endorsers
initializeExchangePoolUser	sameOrgEndorser
createExchangePoolAccounts	sameOrgEndorser
addConversionRate	sameOrgEndorser
updateConversionRate	sameOrgEndorser
mintWithFundingExchangePool	sameOrgEndorser
tokenConversion	endorsers
getConversionRate	endorsers
getConversionHistory	endorsers
getConversionRateHistory	endorsers
getExchangePoolUser	endorsers
getAccountOnHoldBalance	endorsers
getAccountStatus	endorsers
getAccountsByRole	endorsers

Fungible Token Framework Wrapper API Package

Oracle Blockchain Platform Digital Assets Edition includes a wrapper API package that extends the REST API to support operations specific to a deposit token scenario.

The wrapper API package uses the API Gateway service and OCI Functions to deploy API routes specifically designed for the deposit token application. The fungible token framework wrapper API package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `DepositTokenWrapperAPI.zip`, an archive file that contains the wrapper API package including the Terraform scripts required for deployment. You deploy this file to a Resource Manager stack on Oracle Cloud Infrastructure (OCI) to create the necessary Oracle resources for the Wrapper APIs.
- `DepositToken_WrapperAPI.postman_collection.json`, a Postman collection that enables you to test the deployed wrapper APIs. The collection includes pre-configured requests with endpoints and payloads that correspond to the APIs defined in the wrapper API package.

Wrapper APIs

`activateAccount`
Original method name: `activateAccount`

This POST method activates a token account. This method can be called only by a `Token Admin` or the `Org Admin` of the specified organization. For any accounts created prior to the account status functionality, you must call this method to see the account status to active.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"db0738d4a44f6d9c80b24fce7c518c07023f7be19edaa69b272eaf7886b4b925",
        "payload": {
            "assetType": "oaccountStatus",
            "status_id":
"oaccountStatus~d5814d96d8517ac31727d60aace0519c58a425892ab0d378fcfb0a35771f65
ae",
            "account_id":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
            "status": "active"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 194
    }
}
```

addTokenAdmin
Original method name: `addTokenAdmin`

This POST method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "User (Org_Id: CB, User_Id: cb) is already Token Admin."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

addTokenAuditor
This POST method adds a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a message that includes details of the user who was added as a `Token Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"cd81f6c4c9e7c18ece357dbf5c139ef66ef2d6566be3b14de5f6d0a3fd4bb2f0",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 196
    }
}
```

`addOrgAdmin`
Original method name: `addOrgAdmin`

This POST method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization..

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a message that includes details of the user who was added as a `Org Admin` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"96a84dffcb9156f7271dfb414e8c43b540595044cf9145f5fd56e9873797fc4a",
        "payload": {
            "msg": "Successfully added Org Admin (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 197
    }
}
```

addOrgAuditor
This POST method adds a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a message that includes details of the user who was added as a `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"44bbad35a1478cb714e32f7cfd551897868a203520aab9cea5771d3aadc1cf03",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 198
    }
}
```

`addRole`
Original method name: `addRole`

This method adds the role to the specified user and token.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "role": "role value (for example minter / burner / notary)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `role: string` – The name of the role to add to the specified 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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"29ea766dee8e6d273eba3c40a9fea75a1aa85dc3c280d40695f6224c5c52d93c",
        "payload": {
            "msg": "Successfully added role 'notary' to Account Id:
oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8
(Org-Id: CB, User-Id: manager_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 201
    }
}
```

approveMint
Original method name: `approveMint`

Notaries can call this POST method to approve a mint request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – The unique operation ID of the mint request to approve.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0",
        "payload": {
            "msg": "Successfully minted 10 tokens to Account Id:
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB, User-Id: creator_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 204
    }
}
```

executeHoldTokens
Original method name: `executeHoldTokens`

Notaries call this method to approve a hold on tokens. The quantity of tokens put on hold previously by the token owner is now transferred to the recipient. If the `quantity` value is less than the actual hold value, the remaining amount is available again to the owner of the token. If the `roles` behavior is specified in the `behaviors` section of the token model and the `notary_role_name` value is set, the caller account must have notary role. Otherwise, any caller with an account can function as a notary.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "quantity": 1,
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – The unique operation ID of the mint request to approve.
- `quantity: number` – The number of held tokens to transfer.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"c1149aaa486abc4931d9024c18dfcb230bb321723d1160b0bf981c0011c4856a",
        "payload": {
            "msg": "Account Id:
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594
(Org-Id: CB, User-Id: issuer_user_cb) is successfully executed '10' tokens
from Operation Id '8e3145'."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 213
    }
}
```

associateTokenToAccount
Original method name: `associateTokenToAccount`

This POST method associates a specified account ID to a specified token. It can be called by a `Token Admin` or `Org Admin`.

Payload:

```
{
    "accountId": "account_id value",
    "tokenId": "{{bc-token-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `accountId: string` – The ID of the account.
- `tokenId: string` – The ID of the token.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"efc7381fb6fc6174a40e83ff5f09d2bbf7f6f490365e3bbf19d5502c2cfec474",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id": 
"oaccount~1c6aa60e220b8fc70caf4cea1ed723ddb193a00321e5e0004def062816b77090",
            "user_id": "cb12",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-11T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 216
    }
}
```

`createAccount`
Original method name: `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. Account IDs are formed by concatenating the asset type and token ID and then creating a SHA-256 hash over a concatenation of the organization ID and user ID. This method can be called only by a `Token Admin` of the chaincode or the `Org Admin` of the specified organization..

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenType": "fungible",
    "dailyLimits": 
"{\"max_daily_amount\":10000,\"max_daily_transactions\":100}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `daily_limits: DailyLimits` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id": 
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
```

```
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 100,
                "daily_transactions": 0,
                "current_date": "2024-12-09T00:00:00.000Z"
            },
            "encode": "JSON",
            "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
            "blockNumber": 188
        }
}
```

`getAccount`
Original method name: `getAccount`

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.

Query:

```
/getAccount?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "status": "active",
            "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON"
    }
}
```

`getAccountBalance`
Original method name: `getAccountBalance`

This GET method returns the current balance for a specified account and token. This method can be called only by a `Token Admin` or the `AccountOwner` of the account.

```
/getAccountBalance?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-
user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Current Balance is: 100",
            "user_balance": 100
        },
        "encode": "JSON"
    }
}
```

`getAccountsByUser`
Original method name: `getAccountsByUser`

This method returns a list of all accounts for a specified user. This method can be called only by a `Token Admin` of the chaincode or the `Account Owner` of the account.

Query:

```
/getAccountsByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 1000,
                "daily_transactions": 0,
                "current_date": "2024-11-20T00:00:00.000Z"
            },
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "USD",
                "token_name": "cbdc",
                "balance": 0,
                "onhold_balance": 0,
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 1000,
                "daily_transactions": 0,
                "current_date": "2024-11-20T00:00:00.000Z"
            },
            {
                "bapAccountVersion": 0,
                "assetType": "oaccount",
                "account_id": 
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
                "user_id": "admin_user_cb",
                "org_id": "CB",
                "token_type": "fungible",
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 100,
                "daily_transactions": 0,
                "current_date": "2024-12-09T00:00:00.000Z"
            }
        ],
        "encode": "JSON"
    }
}
```

getAccountTransactionHistory
Original method name: `getAccountTransactionHistory`

This GET method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
/getAccountTransactionHistory?tokenId={{bc-token-id}}&orgId={{bc-org-
id}}&userId={{bc-user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
```

```
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
        ],
        "encode": "JSON"
    }
}
```

`getAccountTransactionHistoryWithFiltersFromRichHistDB`
Original method name: `getAccountTransactionHistoryWithFiltersFromRichHistDB`

This method returns the account transaction history details from the rich history database. This method can be called only by a `Token Admin` or the `AccountOwner` of the account. Before you can use this method, you must run Oracle Autonomous Database with Oracle REST Data Services (ORDS) and OAuth enabled, as described in Oracle Database View Definitions for Wholesale CBDC.

```
/getAccountTransactionHistoryWithFiltersFromRichHistDB?tokenId={{bc-token-
id}}&orgId={{bc-org-id}}&userId={{bc-user-id}}&customEndpoint=custom_endpoint
value&bearerToken=bearer_token
value&filters={"pageSize":20,"bookmark":"","startTime":"2022-01-16T15:16:36+00
:00","endTime":"2022-01-17T15:16:36+00:00"}
```

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.
- `customEndpoint` – The RESTful service endpoint of the rich history database to fetch the transaction history from.
- `bearerToken` – The token to use to call the RESTful endpoint to ensure that the request is authorized.
- `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. The `StartTime` and `EndTime` properties must be specified in RFC-3339 format.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
        ],
        "encode": "JSON"
    }
}
```

getNetTokens
Original method name: `getNetTokens`

This GET 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. This method can be called only by a `Token Admin` of the chaincode, or an `Org Admin` of the specified organization.

```
/getNetCBDCTokens?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Net supply of token for Token Id: USD is 878 tokens.",
            "quantity": 878
        },
        "encode": "JSON"
    }
}
```

getOnHoldIds
Original method name: `getOnHoldIds`

This GET method returns a list of all of the holding IDs for a specified user and token. This method can be called only by a `Token Admin`, `Org Admin`, or the `AccountOwner` of the account.

```
/getOnHoldIds?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-user-
id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Holding Ids are: 
ohold~cbdc~USD~8e3147,ohold~cbdc~USD~8e315",
            "holding_ids": [
                "ohold~cbdc~USD~8e3147",
                "ohold~cbdc~USD~8e315"
            ]
        },
        "encode": "JSON"
    }
}
```

`getTotalMintedTokens`
Original method name: `getTotalMintedTokens`

This GET 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`.

Query:

```
/getTotalMintedTokens?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "msg": "Total minted token for Token Id: USD is 910 tokens.",
            "quantity": 910
        },
        "encode": "JSON"
    }
}
```

`getUserByAccountId`
Original method name: `getUserByAccountId`

This GET method returns the user details for a specified account. This method can be called by the `Token Admin`, `Token Auditor`, or `Org Auditor`.

Query:

```
/getUserByAccountId?accountId=account_id value
```

Parameters:

- `accountId: string` – The ID of the account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "token_id": "USD",
            "user_id": "admin_user_cb",
            "org_id": "CB"
        },
        "encode": "JSON"
    }
}
```

`getUsersByRole`
Original method name: `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`.

```
/getUsersByRole?tokenId={{bc-token-id}}&role=role value (for example minter /
burner / notary)
```

Parameters:

- `tokenId: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "users": [
                {
                    "token_id": "USD",
                    "user_id": "creator_user_cb",
                    "org_id": "CB"
                },
                {
                    "token_id": "USD",
                    "user_id": "cb4",
                    "org_id": "CB"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

`holdTokens`
Original method name: `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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "notaryOrgId": "notary_org_id value",
    "notaryUserId": "notary_user_id value",
    "quantity": 1,
    "timeToExpiration": "time_to_expiration value",
    "infoDetails": "{\"category\":\"category 
value\",\"description\":\"description value\"}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `operationId: string` – A unique ID to identify the hold operation. Typically this ID is passed by the client application.
- `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.
- `notaryOrgId: string` – The membership service provider (MSP) ID of the notary in the current organization.
- `notaryUserId: string` – The user name or email ID of the notary.
- `quantity: number` – The number of tokens to put on hold.

- `timeToExpiration` – 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`.
- `infoDetails: JSON` – The description and category as shown in the following example.

```
{
   "category" : "category input",
   "description" : "description input"
}
```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e575d339299bb98afe83207e749cd07654f209673c84c6973738b6094da33062",
        "payload": {
            "msg": "AccountId 
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594 
(Org-Id: CB , User-Id: issuer_user_cb) is successfully holding 10 tokens"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 211
    }
}
```

`init`
Original method name: `init`

This method is called when the chaincode is deployed. The user information is saved as the `Token Admin` of the chaincode.

Payload:

```
{
    "adminList": "[{\"org_id\":\"{{bc-org-id}}\",\"user_id\":\"{{bc-admin-
user}}\"}]"
}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"fdb7dc89832c8045a333823b77fa24ae628178148dc93b3550040e070d7cd807",
        "payload": "",
        "encode": "UTF-8",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 263
    }
}
```

`initializeDepositToken`
Original method name: `initializeDepositToken`

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.

Payload:

```
{
    "tokenAsset": "{\"token_id\":\"{{bc-token-id}}
\",\"token_desc\":\"token_desc value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a JSON representation of the token asset that was created.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"aa7a4f4cc214e1a041a5a6fb7ca7530f08256559e538c9f9582e6fd12c9e65c8",
        "payload": {
            "assetType": "otoken",
            "events": false,
            "token_id": "t1",
            "token_name": "cbdc",
            "token_desc": "token_desc value",
            "token_standard": "ttf+",
            "token_type": "fungible",
            "token_unit": "fractional",
            "behaviors": [
                "divisible",
                "mintable",
                "transferable",
                "burnable",
                "holdable",
                "roles"
            ],
            "roles": {
                "minter_role_name": "minter",
                "burner_role_name": "burner",
                "notary_role_name": "notary",
                "mint_approver_role_name": "notary",
                "burn_approver_role_name": "notary"
            },
            "mintable": {
                "max_mint_quantity": 1000,
                "mint_approval_required": true
            },
            "burnable": {
                "burn_approval_required": true
            },
            "divisible": {
                "decimal": 2
            },
            "token_to_currency_ratio": 999
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 267
    }
}
```

`getAccountHistory`
Original method name: `getAccountHistory`

This GET method returns account history details for a specified user and token.

Query:

```
/getAccountHistory?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-
user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {

"trxId":"2gsdh17fff222467e5667be042e33ce18e804b3e065cca15de306f837e416d7c3e",
              "timeStamp":1629718288,
              "value":{
                 "assetType":"oaccount",

"account_id":"oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d5
46d64f368642f622f",
                 "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~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d5
46d64f368642f622f",
                 "user_id":"user1",
                 "org_id":"Org1MSP",
                 "token_id":"digiCurr101",
                 "token_name":"digicur",
                 "balance":0,
                 "onhold_balance":0,
                 "bapAccountVersion": 0
              }
           }
        ],
        "encode": "JSON"
    }
}
```

`rejectMint`
Original method name: `rejectMint`

This method can be called by a notary to reject a minting request.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "sameOrgEndorser": true
}
```

Parameters:

- `token_id: string` – The ID of the token to reject minting.
- `operation_id: string` – The unique operation ID that represents the mint request.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22",
        "payload": {
            "msg": "Successfully rejected mint request with Operation Id
'op1234' to mint 10 tokens of token id USD"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 205
    }
}
```

`releaseHoldTokens`
Original method name: `releaseHoldTokens`

This POST method releases a hold on tokens. The transfer is not completed and all held tokens are available again to the original owner.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "endorsers": {{endorsers}}
}
```

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.

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"f04ba8895d52bc636d843f88476002bc99d01480c36be87c8fa259cd47a29380",
        "payload": {
            "msg": "Successfully released '10' tokens from Operation Id
'8e3144' to Account Id:
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594
(Org-Id: CB, User-Id: issuer_user_cb)."
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 214
    }
}
```

`removeTokenAdmin`
Original method name: `removeTokenAdmin`

This POST method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. An admin cannot remove themselves.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
```

```
"6a3b9b568d04b5beb29830f91efe4e8c6310b6cf36940cecfb4ab690fbfde739",
        "payload": {
            "msg": "Successfully removed Token Admin (Org_Id: CB, User_Id: 
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 218
    }
}
```

`removeTokenAuditor`
Original method name: `removeTokenAuditor`

This POST method removes a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"a886a6040fbc76374a3c78c89ab0ffc9f7b8391cc5239b169bf3b878cf40c67b",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id: 
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 219
    }
}
```

`removeOrgAdmin`
Original method name: `removeOrgAdmin`

This POST method removes a user as a `Org Admin` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"e2a634f6093f89b1984e20ff86a513fabb7c3ade7cc9e27d9734b4aaf6c88597",
        "payload": {
            "msg": "Successfully removed Org Admin (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 220
    }
}
```

`removeOrgAuditor`
Original method name: `removeOrgAuditor`

This POST method removes a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"c3bc720461004a53b37c68d4bb264858b88d980bc093a0a3ebb62a32974fb306",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 221
    }
}
```

`removeRole`
Original method name: `removeRole`

This method removes a role from a specified user. This method can be called only by a `Token Admin` of the chaincode or an `Org Admin` of the specified organization.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "role": "role value (for example minter / burner / notary)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: 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.
- `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.

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"274f0d0a2c4c3929817fb85b2e857519695c3c238ccf9903b084b87e9be7ee12",
        "payload": {
            "msg": "Successfully removed role 'notary' from Account Id:
oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8
(Org-Id: CB, User-Id: manager_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 200
    }
}
```

`requestMint`
Original method name: `requestMint`

This method can be called by a minter to send a request to the notary to create a specified amount of tokens.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "operationId": "operation_id value",
    "notaryOrgId": "notary_org_id value",
    "notaryUserId": "notary_user_id value",
    "quantity": 1,
    "timeToExpiration": "time_to_expiration value",
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to mint.
- `operationId: string` – The unique operation ID that represents the mint request.
- `notaryOrgId: string` – The membership service provider (MSP) ID of the minter notary who will process the request.
- `notaryUserId: string` – The user name or email ID of the minter notary who will process the request.
- `quantity: number` – The amount of tokens to mint.

- `timeToExpiration` – The time after which the minting request expires and is no longer valid.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e",
        "payload": {
            "msg": "AccountId
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB , User-Id: creator_user_cb) has successfully submitted request to
mint 10 tokens"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 202
    }
}
```

burnTokens
Original method name: `burnTokens`

This POST 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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "quantity": 1,
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `quantity` – The number of tokens to burn.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

```
{
     "category" : "category input",
     "description" : "description input"
}
```

- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a success message with the quantity of tokens burned and the account ID.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"aff0b2dbb163ec8076747525db81fbe8f678ac88a277c5f234337f0747eb1a8d",
        "payload": {
            "msg": "Successfully burned 10 tokens from account id:
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB, User-Id: creator_user_cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 96
    }
}
```

`setMaxDailyAmount`
Original method name: `setMaxDailyAmount`

This POST method is used to set the `maxDailyAmount` parameter in the account details for the specified amount. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenId": "{{bc-token-id}}",
    "maxDailyAmount": 1,
    "endorsers": {{endorsers}}
}
```

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.
- `maxDailyAmount: number` – The maximum daily amount value for the specified account, which defines the maximum amount that can be transacted daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"28682e0564e4721b6c1a8ec106f8c5c98319e9439959dbb9f83d8e6f111d9975",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 222
    }
}
```

`setMaxDailyTransactionCount`
Original method name: `setMaxDailyTransactionCount`

This POST method is used to set the `maxDailyTransactions` parameter in the account details for the specified amount. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenId": "{{bc-token-id}}",
    "maxDailyTransactions": 1,
    "endorsers": {{endorsers}}
}
```

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.
- `maxDailyTransactions: number` – The maximum daily amount value for the specified account, which defines the maximum number of transactions allowed per day.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"8b6fb01de697562ee098110054f05d4a314933bd11ef471991cb43e25b68bad9",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "USD",
            "token_name": "cbdc",
            "balance": 0,
            "onhold_balance": 0,
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 1000,
            "daily_transactions": 0,
            "current_date": "2024-11-20T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 223
    }
}
```

`suspendAccount`
Original method name: `suspendAccount`

This method suspends a fungible token account. It throws an error if an `accountStatus` value is not found in ledger. This method can be called only by a `Token Admin` of the chaincode or by an `Org Admin` of the specified organization.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
            "assetType": "oaccountStatus",
            "status_id":
"oaccountStatus~d5814d96d8517ac31727d60aace0519c58a425892ab0d378fcfb0a35771f65
ae",
            "account_id":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
            "status": "suspended"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 195
    }
}
```

`transferTokens`
Original method name: `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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "quantity": 1,
    "infoDetails": "{\"category\":\"category
value\",\"description\":\"description value\"}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver (payee) in the current organization.
- `toUserId: string` – The user name or email ID of the receiver.
- `quantity: number` – The number of tokens to transfer.
- `infoDetails: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"d613b2494b965811b2fa2106152b7085f2d6d7d43e949b10b8668722d3636fe7",
        "payload": {
            "msg": "Successfully transferred 10 tokens from account id:
oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0
(Org-Id: CB, User-Id: creator_user_cb) to account id:
oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594
(Org-Id: CB, User-Id: issuer_user_cb). Only 999 number of transactions and
```

```
1990 amount transfer left for today: 12/11/2024"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 224
    }
}
```

initializeExchangePoolUser
Original method name: `initializeExchangePoolUser`

This method initializes the exchange pool user, which is a one-time activity. This method can be called only by the `Token Admin`.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
            "assetType": "oconversion",
            "convertor_id":
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
            "org_id": "Org1MSP",
            "user_id": "exchangepooluser"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 196
    }
}
```

createExchangePoolAccounts
Original method name: `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.

Payload:

```
{
    "tokenIds": "[{{bc-token-id}}]",
    "sameOrgEndorser": true
}
```

Parameters:

- `token_ids: string []` – An array of token IDs.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": [
            {
                "account_id": 
"oaccount~cc9d84f6d4a5976532493ef5200c9603e138adc35166ffd5fd1aad9c1647f034",
                "token_id": "USD",
                "status": "created"
            },
            {
                "account_id": 
"oaccount~3d4933111ec8bd6cc1ebb43f2b2c390deb929cfa534f9c6ada8e63bac04a13c0",
                "token_id": "INR",
                "status": "created"
            }
        ],
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 197
    }
}
```

`addConversionRate`
Original method name: `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.

Payload:

```
{
    "fromTokenId": "from_token_id value",
    "toTokenId": "to_token_id value",
    "tokenConversionRate": 10,
    "sameOrgEndorser": true
}
```

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 `fromTokenId` token to the `toTokenId` token.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
          "assetType": "oconversionRate",
          "conversion_rate_id": 
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d578
73b",
          "from_token_id": "USD",
          "to_token_id": "INR",
          "conversion_rate": 10
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 189
    }
}
```

`updateConversionRate`
Original method name: `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.

Payload:

```
{
    "fromTokenId": "from_token_id value",
    "toTokenId": "to_token_id value",
    "tokenConversionRate": 20,
    "sameOrgEndorser": true
}
```

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 `fromTokenId` token to the `toTokenId` token.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
          "assetType": "oconversionRate",
          "conversion_rate_id": 
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d578
73b",
          "from_token_id": "USD",
          "to_token_id": "INR",
          "conversion_rate": 20
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 208
    }
}
```

`mintWithFundingExchangePool`
Original method name: `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.

```
{
    "tokenId": "{{bc-token-id}}",
    "tokenQuantity": 100,
    "percentageTokenToExchangePool": 20,
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to mint.
- `tokenQuantity: number` – The total number of tokens to mint.
- `percentageTokenToExchangePool: number` – The percentage of minted tokens to transfer to the exchange pool token account.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
            "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) "
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 209
    }
}
```

`tokenConversion`
Original method name: `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.

```
{
    "fromTokenId": "from_token_id value",
    "toTokenId": "to_token_id value",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "tokenQuantity": 5,
    "endorsers": {{endorsers}}
}
```

Parameters:

- `fromTokenId: string` – The ID of the token to convert from.
- `toTokenId: string` – The ID of the token to convert to.
- `toOrgId: string` – The membership service provider (MSP) ID of the user in the current organization to receive the tokens.
- `toUserId: string` – The user name or email ID of the user to receive the tokens.
- `tokenQuantity: number` – The total number of tokens to transfer.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"e21d91d98091df77f90105d84074c0b802b01fc97a6b1304247774397fed1294",
        "payload": {
            "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"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 213
    }
}
```

`getConversionRate`
Original method name: `getConversionRate`

This GET method gets the current conversion rate for a pair of tokens. This method can be called by the `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

Query:

```
/getConversionRate?fromTokenId=from_token_id value&toTokenId=to_token_id value
```

Parameters:

- `fromTokenId: string` – The ID of the token to convert from.
- `toTokenId: string` – The ID of the token to convert to.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
          "assetType": "oconversionRate",
          "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d578
73b",
          "from_token_id": "USD",
          "to_token_id": "INR",
          "conversion_rate": 20
        },
        "encode": "JSON"
    }
}
```

getConversionHistory
Original method name: `getConversionRate`

This GET method returns the token conversion history for a specified token account. This method can be called by the `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or the token account owner.

Query:

```
/getConversionHistory?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-
user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
          {
            "transaction_id":
"otransaction~34edd19e03ec8bbbc77bc3372081410a824a5c10f9aa522b3a6390d7e8cb11cf
",
            "from_account_id":
"oaccount~abc74791148b761352b98df58035601b6f5480448ac2b4a3a7eb54bdbebf48eb",
            "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"
          }
        ],
        "encode": "JSON"
    }
}
```

getConversionRateHistory
Original method name: `getConversionRate`

This method returns the token conversion rate history for a pair of tokens. This method can be called by the `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, and by any token account owner.

```
/getConversionRateHistory?fromTokenId=from_token_id
value&toTokenId=to_token_id value
```

Parameters:

- `fromTokenId: string` – The ID of the token to convert from, for the purpose of calculating the conversion rate.
- `toTokenId: 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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  [
          {
            "trxId":
"0b1ba7bc2620e1438b6580365e5c0ab852247ccfa5a3eb2157d3baca02c0e521",
            "timeStamp": "2022-11-30T10:23:38.000Z",
            "value": {
              "assetType": "oconversionRate",
              "conversion_rate_id":
"oconversionRate~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d578
73b",
              "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~91c7eeb0614e7a50b1d5ecad559fcbc80b94034648bf405c9491dacf8d578
73b",
              "from_token_id": "USD",
              "to_token_id": "INR",
              "conversion_rate": 10
            }
          }
        ],
        "encode": "JSON"
    }
}
```

getExchangePoolUser
Original method name: `getExchangePoolUser`

This GET method returns the organization ID and user ID values for the exchange pool user. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode.

Query:

```
/getExchangePoolUser
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  {
           "assetType": "oconversion",
           "convertor_id": 
"bcb1f3b1442c625d3ce205660c5e717c5858a1fe1e12c325df799a851ceaa09b",
           "org_id": "Org1MSP",
           "user_id": "exchangepooluser"
        },
        "encode": "JSON"
    }
}
```

getAccountOnHoldBalance
Original method name: `getAccountOnHoldBalance`

This GET method returns the current on-hold balance for a specified account and token. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or the `AccountOwner` of the account.

Query:

```
/getAccountOnHoldBalance?tokenId={{bc-token-id}}&orgId={{bc-org-
id}}&userId={{bc-user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  {
            "msg":"Total Holding Balance is: 0","holding_balance":0
        },
        "encode": "JSON"
    }
}
```

getAccountStatus
Original method name: `getAccountStatus`

This GET method retrieves the current status of the token account. This method can be called by the `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or by the token account owner.

Query:

```
/getAccountStatus?tokenId={{bc-token-id}}&orgId={{bc-org-id}}&userId={{bc-
user-id}}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  {
            "assetType": "oaccountStatus",
            "status_id": 
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
            "account_id": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
            "status": "active"
        },
        "encode": "JSON"
    }
}
```

`getAccountsByRole`
Original method name: `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.

Query:

```
/getAccountsByRole?tokenId={{bc-token-id}}&role=role value (for example 
minter / burner / notary)
```

Parameters:

- `tokenId: string` – The ID of the token.
- `role: string` – The name of the role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  {
            "accounts": 
["oaccount~digicur~b4f45440aa2a7942db64443d047027e9d714d62cba5c3d546d64f368642
f622f"]
        },
        "encode": "JSON"
    }
}
```

Non-Fungible Token Framework

The non-fungible token framework uses the extended ERC-721 standard that is supported by Blockchain App Builder.

Non-fungible tokens (NFTs) are unique digital assets that represent ownership of a specific item or piece of content. Unlike fungible tokens, each NFT has distinct properties. Blockchain App Builder extends the ERC-721 standard to support the creation and management of NFTs, enabling developers to tokenize unique real-world assets (RWAs) and native digital assets efficiently.

Example use cases:

Digital Art
Tokenizing artworks to provide artists with a platform to sell and track ownership of their creations.

Collectibles
Creating digital collectibles such as trading cards, gaming items, and music, with verifiable ownership.

Property Records
Representing records as NFTs to simplify the ownership, transfer, and management of property.

Letters of Credit
Representing a letters of credit as NFTs to securely transfer trade finance documents, ensuring authenticity, transparency, and automated completeion of global transactions.

Intellectual Property
Assigning ownership of patents or trademarks to NFTs, facilitating easier licensing and transfer.

The enhanced version of Blockchain App Builder that is included with Oracle Blockchain Platform Digital Assets Edition supports the following functions.

- Unique identifiers: Each NFT is assigned a distinct identifier, ensuring its uniqueness and traceability. Only whole NFT creation is supported, which means each NFT represents a unique, indivisible asset without fractional ownership. For information about fractional NFT support, see Combined Token Framework.
- Metadata association: Attach metadata to tokens to provide detailed information about the asset such as descriptions, images, or external links. Metadata properties are fixed once the NFT is created.
- Custom attributes: Specify additional properties that can be updated by the NFT owner.
- Minting and burning: Manage the life cycle of the tokenized assets by creating (minting) and removing (burning) NFTs.
- Ownership transfer: Transfer ownership of NFTs between parties with secure methods that ensure authenticity and provenance.
- Locking: Lock an NFT in a vault so that it cannot be transferred to or burned by any user.
- Role operations: Assign and enforce roles such as minter, burner to provide specific privileges to any user account.

Oracle Blockchain Platform Digital Assets Edition includes a chaincode package and a wrapper API package for the non-fungible token scenario. The chaincode package includes the

NFT Art Collection Marketplace sample, which illustrates use of the framework. The wrapper API package extends the REST API to support operations specific to the NFT Art Collection Marketplace scenario.

Non-Fungible Token Framework Chaincode Package

The non-fungible token framework uses the extended ERC-721 standard that is supported by Blockchain App Builder.

The NFT Art Collection Marketplace sample illustrates the use of the non-fungible token generic framework, which is based on the extended ERC-721 standard supported by Blockchain App Builder. The sample includes a chaincode to represent 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 chaincode implements the methods that are required for managing the non-fungible token life cycle, including token initialization, account operations, role assignments, minting, transfers, and burning. It also provides notary accounts for approving minting, transfer, and burning operations, and supports compliance through daily limits and auditing procedures. The NFT Art Collection Marketplace sample is designed for chaincode development in TypeScript.

The non-fungible token framework chaincode package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `NFTCollectiblesWithERC721.zip`, an archive file that contains the packaged chaincode for deployment.
- `NFTCollectiblesWithERC721-TypeScript.yaml`, a specification file that you can use with Blockchain App Builder to scaffold the `NFTCollectiblesWithERC721` chaincode.
- `NFTCollectiblesWithERC721_postman_collection.json`, a Postman collection that enables you to test the APIs in the chaincode.
- `README.md`, a step-by-step guide for working with the chaincode.

To get the fungible token framework, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Non-Fungible Token Framework**.

For more details about using Postman collections, see the following topics.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Specification File

The specification file that is used to generate the art collection marketplace chaincode includes the `events` attribute. The chaincode events function supports event callbacks in generated chaincodes to enable real-time notifications and trigger workflows. For more information about specification files and the parameters used in specification files, see Input Specification File for Non-Fungible Tokens in *Using Oracle Blockchain Platform*.

The art collection marketplace chaincode is based on the extended ERC-721 standard, as shown in the following specification file.

```
#
# Copyright (c) 2024, Oracle and/or its affiliates. All rights reserved.
#
```

```
# This specification file is an example how to build any whole non-fungible
token application.
# For a whole non-fungible token application, art collection marketplace has
been used as an example.
# Art collection marketplace is a digital marketplace that uses NFTs to
enable the buying, selling, and showcasing of unique art pieces, providing
secure ownership, provenance tracking, and exclusive rights for artists and
collectors.

assets:
    - name: ArtCollection #Asset name
      type: token #Asset type
      symbol: ART         # Token symbol
      standard: erc721+   # Token standard
      events: true # Supports event code generation for non-GET methods

      anatomy:
          type: nonfungible # Token type
          unit: whole  #Token unit

      behavior:
        - indivisible
        - singleton
        - mintable:
            max_mint_quantity: 20000
        - transferable
        - 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: float

          - name: On_Sale_Flag # Custom asset attribute to maintain non-
fungible token selling status in the marketplace
            type: boolean

      metadata: # To maintain the metadata on-chain, this tag will be used.
Users won't be able to update the metadata attribute values after an NFT is
minted.

          - name: Painting_Name # Custom asset attribute to represent the
title given to a piece of artwork.
            type: string

          - name: Description # Custom asset attribute to represent a
detailed explanation or interpretation of the painting's concept, style, or
message.
            type: string

          - name: Painter_Name # Custom asset attribute to represent the name
of the artist who created the painting.
            type: string
```

```
customMethods:
    - executeQuery
    - "post(token_id: string, selling_price: number)" # Post the non-fungible
token for selling in the marketplace.
    - "buy(from_org_id: string, from_user_id: string, to_org_id: string,
to_user_id: string, nonfungible_token_id: string, amount_paid: number)"  #
Buy the non-fungible token after paying the amount using any payment gateways.
```

Endorser Details in Chaincode Methods

Oracle Blockchain Platform Digital Assets Edition adds an endorsement parameter to the request payload for all setter methods. The value of the parameter is either `endorsers` or `sameOrgEndorser`. If the `sameOrgEndorser` parameter is true, transaction endorsements must be from the same organization as the requester. The `endorsers` parameter specifies a list of peers that must endorse the transaction. For more information, see Endorsement Support in Postman Collections. The following table shows the endorser type for each method.

Method	Endorser Type
`activateAccount`	`endorsers`
`addTokenAdmin`	`sameOrgEndorser`
`addRole`	`endorsers`
`addTokenSysRole`	`endorsers`
`balanceOf`	`endorsers`
`buy`	`endorsers`
`createAccount`	`endorsers`
`createArtCollectionToken`	`sameOrgEndorser`
`deleteAccount`	`endorsers`
`getAccountByUser`	`endorsers`
`getAccountHistory`	`endorsers`
`getAccountsByRole`	`endorsers`
`getAccountsByTokenSysRole`	`endorsers`
`getAccountStatus`	`endorsers`
`getAccountStatusHistory`	`endorsers`
`getAccountTransactionHistory`	`endorsers`
`getAccountTransactionHistoryWithFilters`	`endorsers`
`getAllAccounts`	`endorsers`
`getAllLockedNFTs`	`endorsers`
`getAllTokenAdmins`	`endorsers`
`getAllTokens`	`endorsers`
`getAllTokensByUser`	`endorsers`
`getLockedNFTsByOrg`	`endorsers`
`getTokenById`	`endorsers`
`getTokenHistory`	`endorsers`
`getTransactionById`	`endorsers`
`getUserByAccountId`	`endorsers`
`getUsersByRole`	`endorsers`

Method	Endorser Type
`getUsersByTokenSysRole`	`endorsers`
`init`	`endorsers`
`isInRole`	`endorsers`
`isInTokenSysRole`	`endorsers`
`isNFTLocked`	`endorsers`
`isTokenAdmin`	`endorsers`
`lockNFT`	`sameOrgEndorser`
`name`	`endorsers`
`ownerOf`	`endorsers`
`post`	`endorsers`
`removeRole`	`endorsers`
`removeTokenAdmin`	`sameOrgEndorser`
`removeTokenSysRole`	`endorsers`
`safeTransferFrom`	`endorsers`
`suspendAccount`	`endorsers`
`symbol`	`endorsers`
`tokenURI`	`endorsers`
`totalNetSupply`	`endorsers`
`totalSupply`	`endorsers`
`transferFrom`	`endorsers`
`transferTokenSysRole`	`endorsers`
`updateArtCollectionToken`	`sameOrgEndorser`

Non-Fungible Token Framework Wrapper API Package

Oracle Blockchain Platform Digital Assets Edition includes a wrapper API package that extends the REST API to support operations specific to a collectible NFT marketplace.

The wrapper API package uses the API Gateway service and OCI Functions to deploy API routes specifically designed for the collectible marketplace application. The non-fungible token framework wrapper API package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `NFTCollectiblesWithERC721WrapperAPI.zip`, an archive file that contains the wrapper API package including the Terraform scripts required for deployment. You deploy this file to a Resource Manager stack on Oracle Cloud Infrastructure (OCI) to create the necessary Oracle resources for the Wrapper APIs.
- `NFTCollectiblesWithERC721_WrapperAPI.postman_collection.json`, a Postman collection that enables you to test the deployed wrapper APIs. The collection includes pre-configured requests with endpoints and payloads that correspond to the APIs defined in the wrapper API package.

Wrapper APIs

`activateAccount`
Original method name: `activateAccount`

This POST method activates a token account. This method can be called only by an admin. For any accounts created prior to the account status functionality, you must call this method to see the account status to active.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON representation of the account status object for the specified token account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"db0738d4a44f6d9c80b24fce7c518c07023f7be19edaa69b272eaf7886b4b925",
        "payload": {
            "assetType": "oaccountStatus",
            "status_id":
"oaccountStatus~d5814d96d8517ac31727d60aace0519c58a425892ab0d378fcfb0a35771f65
ae",
            "account_id":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
            "status": "active"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 194
    }
}
```

`addTokenAdmin`
Original method name: `addTokenAdmin`

This POST method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a message that includes details of the user who was added as a `Token Admin` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg":"Successfully added Admin (orgId: Org1MSP, userId: User1)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 190
    }
}
```

`addRole`
Original method name: `addRole`

This method adds the role to the specified user and token.

Payload:

```
{
    "role": "role value (for example minter / burner)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `role: string` – The name of the role to add to the specified 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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Successfully added role 'minter' to Account Id:
oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d
(Org-Id: Org1MSP, User-Id: admin)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

addTokenSysRole
Original method name: `addTokenSysRole`

This method adds `Org Admins` to the token chaincode.

Payload:

```
{
    "role": "role value (for example vault)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `role: string` – The name of the role to add to the specified 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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"96a84dffcb9156f7271dfb414e8c43b540595044cf9145f5fd56e9873797fc4a",
        "payload": {
            "msg": "Successfully added Org Admin (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 197
    }
}
```

`balanceOf`
Original method name: `balanceOf`

This GET 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.

Query:

```
/balanceOf?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload":  {
            "totalNfts": 0
        },
        "encode": "JSON"
    }
}
```

`buy`
Original method name: `buy`

This POST method buys a token that is on sale.

Payload:

```
{
    "fromOrgId": "from_org_id value",
    "fromUserId": "from_user_id value",
    "toOrgId": "to_org_id value",
    "toUserId": "to_user_id value",
    "nonfungibleTokenId": "nonfungible_token_id value",
    "amountPaid": 1,
    "endorsers": {{endorsers}}
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender (owner) in the current organization.
- `fromUserId: string` – The user name or email ID of the sender (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.
- `nonfungibleTokenId: string` – The ID of the token to buy.
- `amountPaid: number` – The price of the token.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Token ID: 'monalisa' has been successfully transferred to 
UserID :oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729
dba"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`createAccount`
Original method name: `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 owns. 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 a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "tokenType": "nonfungible",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "assetType": "oaccount",
            "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
            "bapAccountVersion": 0,
            "userId": "admin",
            "orgId": "Org1MSP",
            "tokenType": "nonfungible",
            "noOfNfts": 0
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`createArtCollectionToken`
Original method name: `createArtCollectionToken`

This POST 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.

Payload:

```
{
    "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"tokenUri\":\"tokenUri value\",\"metadata\":
{\"Painting_Name\":\"Painting_Name value\",\"Description\":\"Description
value\",\"Painter_Name\":\"Painter_Name
value\"},\"Price\":999,\"On_Sale_Flag\":true}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to mint. For more information about the properties of the token asset, see the input specification file.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "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~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
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

deleteAccount
Original method name: `deleteAccount`

This POST method deletes a token account. After an account is deleted, the account is in a final state and cannot be updated or changed to any other state. To delete an account, the account balance must be zero. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
          "assetType": "oaccountStatus",
          "statusId": 
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
          "accountId": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
          "status": "deleted"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`getAccountByUser`
Original method name: `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.

Query:

```
/getAccountByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
          "bapAccountVersion": 0,
          "assetType": "oaccount",
          "status": "active",
          "accountId": 
"oaccount~cc301bee057f14236a97d434909ec1084970921b008f6baab09c2a0f5f419a9a",
          "userId": "idcqa",
          "orgId": "appdev",
          "tokenType": "nonfungible",
          "noOfNfts": 0
        },
        "encode": "JSON"
    }
}
```

getAccountHistory
Original method name: `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.

Query:

```
/getAccountHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": [
            {
                "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~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                    "userId": "admin",
                    "orgId": "Org1MSP",
                    "tokenType": "nonfungible",
                    "noOfNfts": 0
                }
            },
            {
                "trxId": 
"e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c",
                "timeStamp": 1649147442,
                "value": {
                    "assetType": "oaccount",
                    "bapAccountVersion" : 1,
                    "accountId": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                    "userId": "admin",
                    "orgId": "Org1MSP",
                    "tokenType": "nonfungible",
                    "noOfNfts": 1
                }
            },
            {
                "trxId":
"d2d1f9c898707ae831e9361bc25da6369eac37b10c87dc04d18d6f3808222f08",
                "timeStamp": 1649137534,
                "value": {
                    "assetType": "oaccount",
                    "bapAccountVersion" : 0,
                    "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                    "userId": "admin",
                    "orgId": "Org1MSP",
                    "tokenType": "nonfungible",
                    "noOfNfts": 0
                }
            }
        ],
        "encode": "JSON"
    }
}
```

getAccountsByRole
Original method name: `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.

Query:

```
/getAccountsByRole?role=role value (for example minter / burner)
```

Parameters:

- `role: string` – The name of the role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "accounts": [

"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d"
            ]
        },
        "encode": "JSON"
    }
}
```

getAccountsByTokenSysRole
Original method name: `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.

Query:

```
/getAccountsByTokenSysRole?role=role value (for example vault)
```

Parameters:

- `role: string` – The name of the `TokenSys` role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "accountIds": [

"oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba"
            ]
        },
        "encode": "JSON"
    }
}
```

`getAccountStatus`
Original method name: `getAccountStatus`

This GET method retrieves 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.

Query:

```
/getAccountStatus?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "assetType": "oaccountStatus",
            "statusId": 
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
            "accountId": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
            "status": "active"
        },
        "encode": "JSON"
    }
}
```

getAccountStatusHistory
Original method name: `getAccountStatusHistory`

This GET method retrieves the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

Query:

```
/getAccountStatusHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
          {
            "trxId": 
"d5c6d6f601257ba9b6edaf5b7660f00adc13c37d5321b8f7d3a35afab2e93e63",
            "timeStamp": "2022-12-02T10:39:14.000Z",
            "value": {
              "assetType": "oaccountStatus",
              "statusId": 
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
              "accountId": 
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
              "status": "suspended"
            }
            },
            {
              "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
              "timeStamp": "2022-12-02T10:37:50.000Z",
              "value": {
                "assetType": "oaccountStatus",
                "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
                "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
                "status": "active"
              }
            }
          ],
          "encode": "JSON"
    }
}
```

getAccountTransactionHistory
Original method name: `getAccountTransactionHistory`

This GET method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

```
/getAccountTransactionHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4
",
                "timestamp": "2022-04-05T09:30:44.000Z",
                "tokenId": "monalisa1",
                "noOfNfts": 1,
                "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "BURN"
            },
            {
                "transactionId": 
"otransaction~a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3ecf53a3
",
                "timestamp": "2022-04-05T09:30:22.000Z",
                "tokenId": "monalisa1",
                "transactedAccount": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "MINT"
            },
            {
                "transactionId": 
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e7
",
                "timestamp": "2022-04-05T09:28:30.000Z",
                "tokenId": "monalisa",
                "transactedAccount": 
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
                "transactionType": "CREDIT"
            },
            {
                "transactionId": 
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9cb
",
                "timestamp": "2022-04-05T09:05:45.000Z",
                "tokenId": "monalisa",
                "transactedAccount": 
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
                "transactionType": "DEBIT"
            },
            {
                "transactionId": 
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c
",
                "timestamp": "2022-04-05T08:30:42.000Z",
                "tokenId": "monalisa",
                "transactedAccount": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "MINT"
            }
        ],
        "encode": "JSON"
    }
}
```

getAccountTransactionHistoryWithFilters
Original method name: `getAccountTransactionHistoryWithFilters`

This GET 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.

```
/getAccountTransactionHistoryWithFilters?orgId={{bc-org-id}}&userId={{bc-user-
id}}&filters={"pageSize":20,"bookmark":"","startTime":"2022-01-16T15:16:36+00:
00","endTime":"2022-01-17T15:16:36+00:00"}
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4
",
                "timestamp": "2022-04-05T09:30:44.000Z",
                "tokenId": "monalisa1",
                "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "BURN"
            },
            {
                "transactionId":
"otransaction~a605f1fa62e511c2945fce5437f983a5e70ec814b82520d3ecd2d81e3ecf53a3
",
                "timestamp": "2022-04-05T09:30:22.000Z",
                "tokenId": "monalisa1",
                "transactedAccount":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "MINT"
            },
            {
                "transactionId":
"otransaction~ca4c07bf04240345de918cbf1f4f3da4b4d0ab044c5b8bea94343e427d9ed4e7
",
                "timestamp": "2022-04-05T09:28:30.000Z",
                "tokenId": "monalisa",
                "transactedAccount":
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
                "transactionType": "CREDIT"
            },
            {
                "transactionId": 
"otransaction~cfb52ffc8c34c7fd86210fcf8c5f53d9f92a056c45ed3a33671d638020c1f9cb
",
                "timestamp": "2022-04-05T09:05:45.000Z",
                "tokenId": "monalisa",
                "transactedAccount": 
"oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba",
                "transactionType": "DEBIT"
            },
            {
                "transactionId": 
"otransaction~e7747b3001a170f88688620956320e9402e1dd8edad8afb4818a08a34647337c
",
                "timestamp": "2022-04-05T08:30:42.000Z",
                "tokenId": "monalisa",
                "transactedAccount": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "transactionType": "MINT"
            }
        ],
        "encode": "JSON"
    }
}
```

`getAllAccounts`
Original method name: `getAllAccounts`

This GET method returns details of all user accounts. This method can be called only by a `Token Admin` of the chaincode.

Query:

```
/getAllAccounts
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                "valueJson": {
                    "assetType": "oaccount",
                    "accountId": 
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                    "userId": "admin",
                    "orgId": "Org1MSP",
                    "tokenType": "nonfungible",
                    "noOfNfts": 1
                }
            }
        ],
        "encode": "JSON"
    }
}
```

`getAllLockedNFTs`
Original method name: `getAllLockedNFTs`

This GET method returns a list of all locked NFTs. This method can be called only by a `Token Admin` of the chaincode or by the Vault Manager (the user with the `TokenSys vault` role).

Query:

```
/getAllLockedNFTs
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "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~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6
a7733463",
                 "creationDate":"2023-10-20T10:26:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a773
3463",
                 "isBurned":false,
                 "isLocked":true,
                 "tokenUri":"token1.example.com",
                 "price":120,
                 "on_sale_flag":false
              }
           }
        ],
        "encode": "JSON"
    }
}
```

getAllTokenAdmins
Original method name: `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.

Query:

```
/getAllTokenAdmins
```

Parameters:

- none

Returns:

- On success, an `admins` array in JSON format that contains `orgId` and `userId` objects.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "admins":[
                {
                    "orgId":"Org1MSP",
                    "userId":"admin"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

`getAllTokens`
Original method name: `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.

Query:

```
/getAllTokens
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "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
                }
            }
        ],
        "encode": "JSON"
    }
}
```

`getAllTokensByUser`
Original method name: `getAllTokensByUser`

This GET 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.

Query:

```
/getAllTokensByUser?orgId={{bc-org-id}}&userId={{bc-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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "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
                }
            }
        ],
        "encode": "JSON"
    }
}
```

getLockedNFTsByOrg
Original method name: `getLockedNFTsByOrg`

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 a `Token Admin` of the chaincode or by the vault manager (the user with the `TokenSys vault` role).

Query:

```
/getLockedNFTsByOrg?orgId={{bc-org-id}}&userId={{bc-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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "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~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6
a7733463",
                        "creationDate":"2023-10-20T10:26:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a773
3463",
                        "isBurned":false,
                        "isLocked":true,
                        "tokenUri":"token1.examplecom",
                        "price":120,
                        "on_sale_flag":false
                    }
                }
            ],
            "encode": "JSON"
    }
}
```

getTokenById
Original method name: `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.

Query:

```
/getTokenById?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token to get.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "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~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
        },
        "encode": "JSON"
    }
}
```

getTokenHistory
Original method name: `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.

```
/getTokenHistory?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "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~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
                }
            },
            {
                "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~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
                }
            },
            {
                "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
                }
            }
        ]
        "encode": "JSON"
```

```
    }
}
```

getTransactionById
Original method name: `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.

Query:

```
/getTransactionById?transactionId=transactionId value
```

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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4
",
            "history": [
                {
                    "trxId":
"6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4",
                    "timeStamp": 1649151044,
                    "value": {
                        "assetType": "otransaction",
                        "transactionId":
"otransaction~6ffd0d94f234c12444a5d5aa559563b59dff4d2280b573fea956dc632bdaf5d4
",
                        "tokenId": "monalisa1",
                        "fromAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                        "toAccountId": "",
                        "triggeredByAccountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
                        "transactionType": "BURN",
                        "timestamp": "2022-04-05T09:30:44.000Z",
                    }
                }
            ]
        },
        "encode": "JSON"
    }
}
```

`getUserByAccountId`
Original method name: `getUserByAccountId`

This GET method returns the user details for a specified account.

Query:

```
/getUserByAccountId?accountId=accountId value
```

Parameters:

- `accountId: string` – The ID of the account.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
          "userId": "admin",
          "orgId": "Org1MSP"
        },
        "encode": "JSON"
    }
}
```

`getUsersByRole`
Original method name: `getUsersByRole`

This method returns a list of all users for a specified role.

```
/getUsersByRole?role=role value (for example minter / burner)
```

Parameters:

- `role: string` – The name of the role to search for.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "users": [
                {
                    "userId": "admin",
                    "orgId": "Org1MSP"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

init
Original method name: `init`

This method is called when the chaincode is deployed. The user information is saved as the `Token Admin` of the chaincode.

Payload:

```
{
    "adminList": "[{\"orgId\":\"{{bc-org-id}}\",\"userId\":\"{{bc-admin-user}}
\"}]"
}
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"fdb7dc89832c8045a333823b77fa24ae628178148dc93b3550040e070d7cd807",
        "payload": "",
        "encode": "UTF-8",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 263
    }
}
```

isInRole
Original method name: `isInRole`

This GET 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.

Query:

```
/isInRole?orgId={{bc-org-id}}&userId={{bc-user-id}}&role=role value (for
example minter / burner)
```

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.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "result":"true"
        },
        "encode": "JSON"
    }
}
```

`isInTokenSysRole`
Original method name: `isInTokenSysRole`

This GET 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 or the `Account Owner` of the account.

Query:

```
/isInTokenSysRole?orgId={{bc-org-id}}&userId={{bc-user-id}}&role=role value
(for example vault)
```

Parameters:

- `role: string` – The name of the `TokenSys` role to search for.
- `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:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "result": true,
            "msg": "Account Id
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba
(Org-Id: Org1MSP, User-Id: user1) has vault role"
        },
        "encode": "JSON"
    }
}
```

`isNFTLocked`
Original method name: `isNFTLocked`

This GET method returns a Boolean value to indicate if a specified token is locked. This method can be called only by a `Token Admin` of the chaincode, the token owner, or the vault manager (the user with the `TokenSys vault` role).

Query:

```
/isNFTLocked?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
           "isNFTLocked":true
        },
        "encode": "JSON"
    }
}
```

`lockNFT`
Original method name: `lockNFT`

This POST 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.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenId: string` – The ID of the token to lock.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
           "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~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6
a7733463",
           "creationDate":"2023-10-20T10:26:29.000Z",

"owner":"oaccount~208e3345ac84b4849f0d2648b2f2f018019886a1230f99304ebff1b6a773
3463",
           "isBurned":false,
           "isLocked":true,
           "tokenUri":"token1.example.com",
           "price":120,
           "on_sale_flag":false
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`name`
Original method name: `name`

This GET method returns the name of the token class. Anyone can call this method.

Query:

```
/name
```

Parameters: None

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {"tokenName": "artcollection"},
        "encode": "JSON"
    }
}
```

`ownerOf`
Original method name: `ownerOf`

This GET method returns the account ID of the owner of the specified token ID. Anyone can call this method.

Query:

```
/ownerOf?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- A JSON object of the owner's account ID.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "owner": 
"oaccount~d6d22c3167e3c6ab9ee5653e1a008c37c20cc47ebb0229ca0aedfafe64c675b8"
        },
        "encode": "JSON"
    }
}
```

`post`
Original method name: `post`

This POST method posts a token for sale for a specified price.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "sellingPrice": 1,
    "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `sellingPrice: number` – The price of the token.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Token ID: 'monalisa'  has been posted for selling in the 
marketplace"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`removeRole`
Original method name: `removeRole`

This method removes a role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "role": "role value (for example minter / burner)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Successfully removed role 'minter' from Account Id:
oaccount~ec32cff8635a056f3dda3da70b1d6090d61f66c6a170c4a95fd008181f729dba
(Org-Id: Org1MSP, User-Id: user1)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`removeTokenAdmin`
Original method name: `removeTokenAdmin`

This POST method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. An admin cannot remove themselves.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Successfully removed Admin (orgId: Org1MSP, userId: 
User1)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

removeTokenSysRole
Original method name: `removeTokenSysRole`

This method removes a `TokenSys` role from a specified user and token. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "role": "role value (for example vault)",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to remove from the specified 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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "msg": "Successfully removed role 'vault' from Account Id: 
oaccount~bf07f584a94be44781e49d9101bfaf58c6fbbe77a4dfebdb83c874c2caf03eba 
(Org-Id: Org1MSP, User-Id: user1)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`safeTransferFrom`
Original method name: `safeTransferFrom`

This is an asynchronous function. This POST 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.

Payload:

```
{
    "fromOrgId": "fromOrgId value",
    "fromUserId": "fromUserId value",
    "toOrgId": "toOrgId value",
    "toUserId": "toUserId value",
    "tokenId": "{{bc-token-id}}",
    "data": "data value",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "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)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

suspendAccount
Original method name: `suspendAccount`

This method suspends a fungible token account. It throws an error if an `accountStatus` value is not found in ledger. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "assetType": "oaccountStatus",
            "statusId":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
            "accountId":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
            "status": "suspended"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

symbol
Original method name: `symbol`

This method returns the symbol of the token class. Anyone can call this method.

Query:

```
/symbol
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "symbol": "PNT"
        },
        "encode": "JSON"
    }
}
```

tokenURI
Original method name: `tokenURI`

This method returns the URI of a specified token. Anyone can call this method.

Query:

```
/tokenURI?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "tokenURI": "https://
bafybeid6pmpp62bongoip5iy2skosvyxh3gr7r2e35x3ctvawjco6ddmsq\.ipfs.infura-
ipfs.io/?filename=MonaLisa.jpeg"
        },
        "encode": "JSON"
    }
}
```

`totalNetSupply`
Original method name: `totalNetSupply`

This GET 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.

```
/totalNetSupply
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "totalNetSupply": 1
        },
        "encode": "JSON"
    }
}
```

`totalSupply`
Original method name: `totalSupply`

This GET method returns the total number of minted tokens. This method can be called only by a `Token Admin` of the chaincode.

Query:

```
/totalSupply
```

Parameters:

- none

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "totalSupply": 3
        },
        "encode": "JSON"
    }
}
```

transferFrom

Original method name: `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.

Payload:

```
{
    "fromOrgId": "fromOrgId value",
    "fromUserId": "fromUserId value",
    "toOrgId": "toOrgId value",
    "toUserId": "toUserId value",
    "tokenId": "{{bc-token-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "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)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`transferTokenSysRole`
Original method name: `transferTokenSysRole`

This method transfers a `TokenSys` role from a user to another user.

Payload:

```
{
    "role": "role value (for example vault)",
    "fromOrgId": "fromOrgId value",
    "fromUserId": "fromUserId value",
    "toOrgId": "toOrgId value",
    "toUserId": "toUserId value",
    "endorsers": {{endorsers}}
}
```

Parameters:

- `role: string` – The name of the `TokenSys` role to transfer.
- `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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "payload": {
            "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)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

`updateArtCollectionToken`
Original method name: `updateArtCollectionToken`

This method updates token properties. 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.

Payload:

```
{
    "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc 
value\",\"tokenUri\":\"tokenUri value\",\"Price\":999,\"On_Sale_Flag\":true}",
    "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to update. For more information about the properties of the token asset, see the input specification file.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"bd7416689b1acdace3c557faebbc0ad9a51671c10278ba6909350a6fe4b08eed",
        "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~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
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 193
    }
}
```

Combined Token Framework

The combined token framework uses the extended ERC-1155 standard that is supported by Blockchain App Builder.

The combined token framework supports both fungible and non-fungible tokens (NFTs) in a single smart contract, so that developers can manage multiple token types efficiently, reducing complexity and improving scalability. Blockchain App Builder extends the ERC-1155 standard to facilitate the development of applications that require a mix of token functionalities. The combined token framework also supports fractional NFTs.

Example use cases:

Gaming Platforms
Issuing in-game currencies (fungible tokens) and unique items such as weapons or skins (NFTs) in the same system.

Event Ticketing and Loyalty Points
Creating event tickets as NFTs with verifiable ownership and streamlining reward accumulation and redemption with fungible tokens.

Real Estate
Managing properties as fractional NFTs to enable shared ownership, liquidity, and transparent transactions on the blockchain.

The enhanced version of Blockchain App Builder that is included with Oracle Blockchain Platform Digital Assets Edition supports the following functions.

- Multiple fungible tokens: Create multiple fungible token types in a single smart contract.
- Multiple token types: Create and manage fungible and non-fungible token types in a single smart contract, streamlining deployment and interaction. One user can have multiple accounts for each fungible token type but only one NFT account.
- Batch operations: Complete multiple token transfers or actions in a single transaction, enhancing efficiency.
- Locking: Lock an NFT in a vault so that it cannot be transferred to or burned by any user.
- Role operations: Assign and enforce roles such as minter, burner to provide specific privileges to any user account.

Oracle Blockchain Platform Digital Assets Edition includes a chaincode package and a wrapper API package for the combined (fungible and non-fungible) token scenario. The chaincode package includes the NFT Art Collection Marketplace sample, which illustrates use of the framework. The wrapper API package extends the REST API to support operations specific to the NFT Art Collection Marketplace scenario.

Combined Token Framework Chaincode Package

The combined token framework uses the extended ERC-1155 standard that is supported by Blockchain App Builder.

The NFT Art Collection Marketplace sample illustrates the use of the combined token generic framework. The sample includes a chaincode to represent a marketplace for buying and selling non-fungible tokens (NFTs) associated with works of art. In this sample, the NFT platform provider onboards museums who can mint (create) NFTs for artworks in the blockchain network. Consumers can then buy NFTs from the museums using Eth coins or ERC-20 coins. When consumers purchase NFTs, museums award loyalty tokens to their fungible token accounts. Consumers can also resell NFTs. The chaincode implements the methods that are required for managing token life cycles, including token initialization, account operations, role assignments, minting, transfers, and burning. It also provides notary accounts for approving minting, transfer, and burning operations, and supports compliance through daily limits and auditing procedures. The NFT Art Collection Marketplace sample is designed for chaincode development in TypeScript.

The combined token framework is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `NFTCollectiblesWithERC1155.zip`, an archive file that contains the packaged chaincode for deployment.
- `NFTCollectiblesWithERC1155-TypeScript.yaml`, a specification file that you can use with Blockchain App Builder to scaffold the `NFTCollectiblesWithERC1155` chaincode.
- `NFTCollectiblesWithERC1155_postman_collection.json`, a Postman collection that enables you to test the APIs in the chaincode.

- `README.md`, a step-by-step guide for working with the chaincode.

To get the fungible token framework, in the Oracle Blockchain Platform Digital Assets console click the **Digital Assets** tab and then select **Combined Token Framework**.

For more details about using Postman collections, see the following topics.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Specification File

The specification file that is used to generate the art collection marketplace chaincode includes the `events` attribute. The chaincode events function supports event callbacks in generated chaincodes to enable real-time notifications and trigger workflows. For more information about specification files and the parameters used in specification files, see Input Specification File for Combined Tokens in *Using Oracle Blockchain Platform*.

The art collection marketplace chaincode is based on the extended ERC-1155 standard, as shown in the following specification file.

```
#
# Copyright (c) 2024, Oracle and/or its affiliates. All rights reserved.
#

# This specification file is an example how to build any whole combined token
application that includes fungible and non-fungible tokens together.
# For a combined token application, art collection marketplace with loyalty
token has been used as an example.
# Art collection marketplace is a digital marketplace for buying and selling
NFT art that rewards users with fungible loyalty tokens for their
participation.

assets:
    - name: ArtCollection #Asset name
      type: token #Asset type
      symbol: ART         # Token symbol
      standard: erc721+   # Token standard
      events: true # Supports event code generation for non-GET methods

      anatomy:
          type: nonfungible # Token type
          unit: whole  #Token unit

      behavior:
        - indivisible
        - singleton
        - mintable:
            max_mint_quantity: 20000
        - transferable
        - 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: float

          - name: On_Sale_Flag # Custom asset attribute to maintain non-
fungible token selling status in the marketplace
            type: boolean

      metadata: # To maintain the metadata on-chain, this tag will be used.
Users won't be able to update the metadata attribute values after an NFT is
minted.

          - name: Painting_Name # Custom asset attribute to represent the
title given to a piece of artwork.
            type: string

          - name: Description # Custom asset attribute to represent a
detailed explanation or interpretation of the painting's concept, style, or
message.
            type: string

          - name: Painter_Name # Custom asset attribute to represent the name
of the artist who created the painting.
            type: string

    - name: Loyalty #Asset name
      type: token #Asset type
      standard: erc1155+   # Token standard
      events: true # Supports event code generation for non-GET methods

      anatomy:
          type: fungible # Token type
          unit: fractional  #Token unit

      behavior: # Token behaviors
        - divisible:
              decimal: 2
        - mintable:
            max_mint_quantity: 100000
        - transferable
        - burnable
        - roles:
            minter_role_name: minter

      properties:
          - name: Token_Name # Custom attribute to represent the token name.
            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

customMethods:
    - executeQuery
    - "post(token_id: string, selling_price: number)" # Post the non-fungible
```

```
token for selling in the marketplace
    - "buyWithEthCoin(from_org_id: string, from_user_id: string, to_org_id:
string, to_user_id: string, nft_id: string[], loyalty_id: string[], eth_qty:
number[], loyalty_reward_quantity: number[])"  # Buy the non-fungible token
after paying the amount using Eth Coin and receive loyalty points in return
```

Combined Token Framework Wrapper API Package

Oracle Blockchain Platform Digital Assets Edition includes a wrapper API package that extends the REST API to support operations specific to a collectible NFT marketplace.

The wrapper API package uses the API Gateway service and OCI Functions to deploy API routes specifically designed for the collectible marketplace application. The non-fungible token framework wrapper API package is downloadable from the Oracle Blockchain Platform console, and includes the following components.

- `NFTCollectiblesWithERC1155WrapperAPI.zip`, an archive file that contains the wrapper API package including the Terraform scripts required for deployment. You deploy this file to a Resource Manager stack on Oracle Cloud Infrastructure (OCI) to create the necessary Oracle resources for the Wrapper APIs.
- `NFTCollectiblesWithERC1155_WrapperAPI.postman_collection.json`, a Postman collection that enables you to test the deployed wrapper APIs. The collection includes pre-configured requests with endpoints and payloads that correspond to the APIs defined in the wrapper API package.

Wrapper APIs

`activateAccount`
Original method name: `activateAccount`

This POST method activates a token account. This method can be called only by an admin or account owner. For existing accounts where an `accountStatus` is not found in the ledger, the method returns an `accountStatus` object with the status set to `active`.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON representation of the updated account status object for the fungible token account.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "active"
}
```

addTokenAdmin
Original method name: `addTokenAdmin`

This POST method adds a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

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)"
}
```

addRole
Original method name: `addRole`

This POST method adds the role to the specified user and token. This method can be called only by a `Token Admin` of the chaincode. The tokenDetails parameter Fungible tokens require the `tokenId` value as input. Non-fungible tokens require the tokenName as input, to achieve that we use tokenDetail parameter to specify details of fungible and nonfungible tokens differently.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "role": "role minter/burner"
    "tokenDetails": "{"tokenName": "token name value"}"
    "endorsers": {{endorsers}}
}
```

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.
- `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"}
  ```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "msg": "Successfully added role 'minter' to Account Id:
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(Org-Id: appdev, User-Id: user2)"
}
```

`associateFungibleTokenToAccount`
Original method name: `associateFungibleTokenToAccount`

This POST method associates a user's fungible token account to a specified token.

Payload:

```
{
    "tokenId": "{{bc-token-id}}",
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "tokenId": "tokenOne"
    }
  ],
  "associatedNftAccount": ""
}
```

`buyWithEthCoin`
Original method name: `buyWithEthCoin`

Any account owner can use this POST method to buy an NFT using Ethereum and to transfer loyalty tokens as reward points.

Payload:

```
{
   "fromOrgId":"from_org_id value",
   "fromUserId":"from_user_id value",
   "toOrgId":"to_org_id value",
   "toUserId":"to_user_id value",
   "nftId":"[\"nft_id value\"]",
   "loyaltyId":"[\"loyalty_id value\"]",
   "ethQty":"[eth_qty value]",
   "loyaltyRewardQuantity":"[loyalty_reward_quantity value]",
   "endorsers":{["endorsers"]}
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender (owner) in the current organization.
- `fromUserId: string` – The user name or email ID of the sender (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.
- `nftId: string` – The ID of the token to buy.
- `loyaltyId: string` – The ID of the fungible token that represents loyalty points.
- `ethQty: number` – The price of the token in Ethereum.
- `loyaltyRewardQuantity: string` – The quantity of loyalty points to transfer.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "msg": "Token ID : 'artcollection1' has been successfully transferred to
UserID : 'user1'"
}
```

`balanceOfBatch`
Original method name: `balanceOfBatch`

This GET method completes a batch operation that gets the balances of token accounts. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

Query:

```
/balanceOfBatch?orgIds=["{{bc-org-id}}"]&userIds=["{{bc-user-
id}}"]&tokenIds=["{{bc-token-id}}"]
```

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:

```
[
    {
        "orgId": "AppBldFFFFMay22",
        "userId": "user2",
        "userAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "tokenAccountId": 
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "tokenId": "FNFT",
        "balance": 100
    },
    {
        "orgId": "AppBldFFFFMay22",
        "userId": "user2",
        "userAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "tokenAccountId": 
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
        "tokenId": "FT",
        "balance": 50
    },
    {
        "orgId": "AppBldFFFFMay22",
        "userId": "example_minter",
        "userAccountId": 
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
        "tokenAccountId": 
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "tokenId": "FNFT",
        "balance": 10
    }
]
```

`batchTransferFrom`
Original method name: `batchTransferFrom`

This POST method completes a batch operation that transfers tokens specified in a list of token IDs from one user to another user. This method can be called by any user.

Payload:

```
{
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "toOrgId": "toOrgId value",
 "toUserId": "toUserId value",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "endorsers": {{endorsers}}
}
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the sender.
- `fromUserId: string` – The user ID of the sender.
- `toOrgId: string` – The membership service provider (MSP) ID of the receiver.
- `toUserId: string` – The user 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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

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: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    }
]
```

`burnBatch`
Original method name: `burnBatch`

This POST method deactivates, or burns, the specified fungible and non-fungible tokens. Any user with the burner role can call this method.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "sameOrgEndorser": true
}
```

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..
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

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: user2)"
  },
  {
    "msg": "Successfully burned 5 tokens of tokenId: tokenOne from Account-ID
oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a
(Org-Id: appdev, User-Id: user2)"
  },
  {
    "msg": "Successfully burned 2 token share of tokenId: FNFT from Account-
ID oaccount~87bcb699d507368ee3966cd03ee6d7736ffc55dde8c0f0e16b14866334ac504a
(Org-Id: AutoF1377358917, User-Id: user2)"
  }
]
```

burnNFT
Original method name: `burnNFT`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenId": "{{bc-token-id}}",
 "sameOrgEndorser": true
}
```

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
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Returns:

- On success, a JSON array that contains a message about the burn operation.

Return Value Example:

```
{
    "msg": "Successfully burned NFT token: 'art' from Account-Id: 
oaccount~76cb672eeb1bd535899562a840d0c15a356db89e24bc8b43ac1dba845a4282c6 
(Org-Id: appdev, User-Id: user2)"
  }
```

`createAccount`
Original method name: `createAccount`

This POST 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~`.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "ftAccount": true,
 "nftAccount": true,
 "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~cf20877546f52687f387e7c91d88b9722c97e1a456cc0484f40c747f7804feae",
  "userId": "user1",
  "orgId": "appdev",
  "totalAccounts": 2,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b",
      "tokenId": ""
    }
  ],
  "associatedNftAccount":
"oaccount~73c3e835dac6d0a56ca9d8def08269f83cefd59b9d297fe2cdc5a9083828fa58"
}
```

`createArtCollectionToken`
Original method name: `createArtCollectionToken`

This POST 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.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"tokenUri\":\"tokenUri value\",\"tokenMetadata\":
{\"Painting_Name\":\"Painting_Name value\",\"Description\":\"Description
value\",\"Painter_Name\":\"Painter_Name
value\"},\"Price\":999,\"On_Sale_Flag\":true}",
 "quantity": 1,
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to mint. For more information about the properties of the token asset, see the input specification file.
- `quantity: number` – The number of tokens to mint. The only supported value for this parameter is `1`.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

`createLoyaltyToken`
Original method name: `createLoyaltyToken`

This POST method creates tokens. Every token that is defined has its own create method. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"Token_Name\":\"Token_Name value\",\"Token_to_Currency_Ratio\":999}",
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset. The properties of the asset are defined in the model file.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "loyalty",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

`createTokenAccount`
Original method name: `createTokenAccount`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenType": "nonfungible",
 "endorsers": {{endorsers}}
}
```

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`.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON object of the token account that was created.

Return Value Example:

```
{
  "assetType": "ouaccount",
  "accountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc",
  "userId": "user2",
  "orgId": "appdev",
  "totalAccounts": 1,
  "totalFtAccounts": 1,
  "associatedFtAccounts": [
    {
      "accountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
      "tokenId": ""
    }
  ],
  "associatedNftAccount":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a"
}
```

`createUserAccount`
Original method name: `createUserAccount`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

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": ""
}
```

deleteAccount
Original method name: `deleteAccount`

This POST method deletes a token account. After an account is deleted, the account is in a final state and cannot be updated or changed to any other state. To delete an account, the account balance must be zero. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Returns:

- On success, a JSON representation of the token account status.

Return Value Example:

```
{
  "assetType": "oaccountStatus",
  "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
  "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
  "status": "deleted"
}
```

exchangeToken

Original method name: `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.

Payload:

```
{
 "fromTokenId": "fromTokenId value",
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "fromTokenQuantity": 1,
 "toTokenId": "toTokenId value",
 "toOrgId": "toOrgId value",
 "toUserId": "toUserId value",
 "toTokenQuantity": 1,
 "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

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: AppBldFFFFMay22, UserId: user2) to 10 tokens of type fungible with
tokenId: [loy1] from Account
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(OrgId: AppBldFFFFMay22, UserId: example_minter)"
}
```

`getAccount`
Original method name: `getAccount`

This GET 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.

Query:

```
/getAccount?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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 that includes token account details.

Return Value Example

```
{
    "assetType": "oaccount",
    "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
    "userId": "user2",
    "orgId": "AppBldFFFFMay22",
    "tokenType": "nonfungible",
    "noOfNfts": 3
}
```

`getAccountDetailsByUser`
Original method name: `getAccountDetailsByUser`

This GET 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.

Query:

```
/getAccountDetailsByUser?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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:

```
{
    "userAccountId":
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
    "associatedFTAccounts": [
        {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "tokenId": "FT",
            "balance": 50
        }
    ],
    "associatedNFTAccount": {
        "accountId":
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "associatedNFTs": [
            {
                "nftTokenId": "FNFT",
                "tokenShare": 100
            },
            {
                "nftTokenId": "FNFT2",
                "tokenShare": 110
            },
            {
                "nftTokenId": "NFT"
            }
        ]
    }
}
```

`getAccountHistory`
Original method name: `getAccountHistory`

This GET method returns account history for a specified token account. For NFT accounts, the `tokenId` parameter must be empty. This method can be called only by the `Token Admin` of the chaincode or by the account owner.

Query:

```
/getAccountHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}&tokenId={{bc-
token-id}}
```

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 fungible token.

Return Value Example:

```
[
    {
        "trxId":
"a2cfc6fc064334d6b9931cdf67193711ec2ff5c50a4714f11855fe7384f00e35",
        "timeStamp": "2023-06-06T14:44:31.000Z",
        "value": {
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "assetType": "oaccount",
            "balance": 100,
            "orgId": "AppBldFFFFMay22",
            "tokenId": "loy1",
            "tokenName": "loyalty",
            "tokenType": "fungible",
            "userId": "user2"
        }
    },
    {
        "trxId":
"de483cf7505ae4e7018c4b604c3ab9327c2fb1f802d9408e22735667c1d6997f",
        "timeStamp": "2023-06-06T14:43:23.000Z",
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "userId": "user2",
            "orgId": "AppBldFFFFMay22",
            "tokenType": "fungible",
            "tokenId": "loy1",
            "tokenName": "loyalty",
            "balance": 0
        }
    },
    {
        "trxId":
"db053e653d3ad9aa5b7b6e04b7cd51aacfbb413272d857a155b60d2a6a12bf4d",
        "timeStamp": "2023-06-05T16:59:08.000Z",
        "value": {
            "assetType": "oaccount",
            "accountId":
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
            "userId": "user2",
            "orgId": "AppBldFFFFMay22",
            "tokenType": "fungible",
            "tokenId": "",
            "balance": 0
        }
    }
]
```

`getAccountStatus`
Original method name: `getAccountStatus`

This GET method retrieves 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.

Query:

```
/getAccountStatus?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "active"
}
```

`getAccountStatusHistory`
Original method name: `getAccountStatusHistory`

This GET method retrieves the history of the account status. This method can be called by the `Token Admin` of the chaincode or by the token account owner.

Query:

```
/getAccountStatusHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "suspended"
    }
  },
  {
    "trxId":
"e6c850cfa084dc20ad95fb2bb8165eef3a3bd62a0ac867cccee57c2003125183",
    "timeStamp": "2022-12-02T10:37:50.000Z",
    "value": {
      "assetType": "oaccountStatus",
      "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
      "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
      "status": "active"
    }
  }
]
```

`getAccountTransactionHistory`
Original method name: `getAccountTransactionHistory`

This GET method returns account transaction history. This method can be called only by a `Token Admin` of the chaincode or by the account owner.

Query:

```
/getAccountTransactionHistory?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddfae140bc
~7c88c736df38d5622512f1e8dcdd50710eb47c953f1ecb24ac44790a9e2f475b",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 10,
        "triggeredByUserAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount": 
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "transactionType": "DEBIT",
        "balance": 90
    },
    {
        "transactionId": 
"otransaction~3a6b23c3003626f3947e990eddbd7ac23398d2200e2eb3eac745e6ddfae140bc
~178e3730bc5bee50d02f1464a4eebf733a051905f651e5789039adb4a3edc114",
        "timestamp": "2023-06-06T14:48:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount": 
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446",
        "transactionType": "DEBIT"
    },
    {
        "transactionId": 
"otransaction~c369929e28e78de06c72d020f1418c9a154a7dd280b2e22ebb4ea4485e249124
~a7cefb22ff39ee7e36967be71de27da6798548c872061a62dabc56d88d50b930",
        "timestamp": "2023-06-06T14:47:08.000Z",
        "tokenId": "NFT",
        "triggeredByUserAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount": 
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT"
    },
    {
        "transactionId": 
"otransaction~114a1bc78d04be48ee6dc140c32c042ee9481cb118959626f090eec744522422
~e4eb15d9354f694230df8835ade012100d82aa43672896a2c7125a86e3048f9f",
        "timestamp": "2023-06-05T17:17:57.000Z",
        "tokenId": "FNFT",
        "transactedAmount": 100,
        "triggeredByUserAccountId": 
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
        "transactedAccount": 
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371",
        "transactionType": "MINT",
        "balance": 100
    }
]
```

`getAccountsByRole`
Original method name: `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.

Query:

```
/getAccountsByRole?role=role value (for example minter /
burner)&tokenDetail={"tokenName":"tokenName value"}
```

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~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
"oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b"
  ]
}
```

`getAllAccounts`
Original method name: `getAllAccounts`

This GET method returns details of all user accounts. This method can be called only by a `Token Admin` of the chaincode.

Query:

```
/getAllAccounts
```

Parameters:

- none

Returns:

- On success, a JSON array of all accounts.

Return Value Example:

```
[
        {
            "assetType": "ouaccount",
            "accountId":
```

```
"ouaccount~412de5e3998dcd100973e1bad6e8729fddc1c7ff610beab8376d733a35c51f38",
            "userId": "user2",
            "orgId": "AppBldFFFFMay22",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId": 
"oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount": 
"oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371"
        },
        {
            "assetType": "ouaccount",
            "accountId": 
"ouaccount~9501bb774c156eb8354dfe489250ea91f757523d70f08ee494bda98bb352003b",
            "userId": "example_minter",
            "orgId": "AppBldFFFFMay22",
            "totalAccounts": 2,
            "totalFtAccounts": 1,
            "associatedFtAccounts": [
                {
                    "accountId": 
"oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c",
                    "tokenId": "loy1"
                }
            ],
            "associatedNftAccount": 
"oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446"
        },
    ]
```

getAllTokenAdmins
Original method name: `getAllTokenAdmins`

This GET 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.

Query:

```
/getAllTokenAdmins
```

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"
    }
  ]
}
```

`getAllTokens`
Original method name: `getAllTokens`

This GET 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.

Query:

```
/getAllTokens
```

Parameters:

- none

Return Value Example:

```
[{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "status value"
}]
```

`getAllTokensByUser`
Original method name: `getAllTokensByUser`

This GET 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.

Query:

```
/getAllTokensByUser?orgId={{bc-org-id}}&userId={{bc-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.

Return Value Example:

```
[{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "status value"
}]
```

`getTokenById`
Original method name: `getTokenById`

This GET 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 response includes the list of token owners.

Query:

```
/getTokenById?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the the token to get.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
```

```
         }
]
```

`getTokenDecimal`
Original method name: `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.

Query:

```
/getTokenDecimal?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "msg": "Token Id: tokenOne has 2 decimal places."
}
```

`getTokenHistory`
Original method name: `getTokenHistory`

This GET method returns the history for a specified token ID. Anyone can call this method.

```
/getTokenHistory?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }
]
```

getTokensByName
Original method name: `getTokensByName`

This GET 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.

Query:

```
/getTokensByName?tokenName=tokenName value
```

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
        }
      }
    }
]
```

getTransactionById
Original method name: `getTransactionById`

This GET method returns transaction history for a specified transaction ID. This is an asynchronous method. This method can be called by any user.

Query:

```
/getTransactionById?transactionId=transactionId value
```

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~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cbe
~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a53",
  "history": [
    {
      "trxId":
"9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cbe",
      "timeStamp": "2022-12-08T09:01:28.000Z",
      "value": {
        "assetType": "otransaction",
        "transactionId":
"otransaction~9ea7b05ab099f7ff4db8342b8c3609031f1d54f11205906e7f1fe88661fe3cbe
~33b59ce0c89e96c1e16449f24301581e8e71954f38ad977c7eb6f065e87f2a53",
        "tokenId": "tokenOne",
        "fromAccountId":
"oaccount~1422a74d262a3a55a37cd9023ef8836f765d0be7b49d397696b9961d7434d22a",
        "toAccountId": "",
        "transactionType": "BURN",
        "amount": 5,
        "timestamp": "2022-12-08T09:01:28.000Z",
        "triggeredByUserAccountId":
"ouaccount~24ffd4d32a028a85b4b960f5d55536c837b5429bc7f346150adfa904ec2937cc"
      }
    }
  ]
}
```

getUserByAccountId
Original method name: `getUserByAccountId`

This GET method returns the organization ID and user ID for a specified account ID.

Query:

```
/getUserByAccountId?accountId=accountId value
```

Parameters:

- `accountId: string` – The ID of the account.

Return Value Example:

```
{
    "orgId": "AppBldFFFFMay22",
    "userId": "user2"
}
```

`getUsersByRole`
Original method name: `getUsersByRole`

This GET method returns a list of all users for a specified role.

Query:

```
/getUsersByRole?role=role value (for example minter /
burner)&tokenDetail={"tokenName":"tokenName value"}
```

Parameters:

- `role: string` – The name of the role to search for.
- `tokenDetail: 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"
            }
        ]
}
```

`init`
Original method name: `init`

This POST method is called when the chaincode is deployed. The user information is saved as the `Token Admin` of the chaincode.

Payload:

```
{
 "adminList": "[{\"orgId\":\"{{bc-org-id}}\",\"userId\":\"{{bc-user-id}}\"}]"
}
```

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.

Returns:

- On success, a message with no payload.

Return Value Example:

```
{

}
```

`isInRole`
Original method name: `isInRole`

This GET 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.

Query:

```
/isInRole?orgId={{bc-org-id}}&userId={{bc-user-id}}&role=role value (for
example minter / burner)&tokenDetail={"tokenName":"tokenName value"}
```

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.

- `tokenDetail: 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: user2) has minter role"
}
```

`isTokenAdmin`
Original method name: `isTokenAdmin`

This GET 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.

Query:

```
/isTokenAdmin?orgId={{bc-org-id}}&userId={{bc-user-id}}
```

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
}
```

`mintBatch`
Original method name: `mintBatch`

This POST 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.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

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: AppBldFFFFMay22, User-Id: user2).",
    "details": [
        {
            "msg": "Successfully minted 100 tokens of fractional tokenId: 
plot55 to Org-Id: AppBldFFFFMay22, User-Id: user2"
        },
        {
            "msg": "Successfully minted 100 tokens of tokenId: loyalty to 
Token-Account-Id 
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e"
        }
    ]
}
```

`name`
Original method name: `name`

This GET method returns the name of the token class. Anyone can call this method.

Query:

```
/name?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "tokenName": "artcollection"
}
```

`ownerOf`
Original method name: `ownerOf`

This GET method returns the account ID of the owners of the specified token ID. Anyone can call this method.

Query:

```
/ownerOf?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Returns:

- A JSON object of the account IDs of the owners.

Return Value Example:

```
[
    {
        "accountId":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
        "orgId": "Org1MSP",
        "userId": "admin"
    },
    {
        "accountId":
"oaccount~74108eca702bab6d8548e740254f2cc7955d886885251d52d065042172a59db0",
        "orgId": "Org1MSP",
        "userId": "user"
    }
]
```

`post`
Original method name: `post`

This POST method posts a token for sale for a specified price.

Payload:

```
{
 "tokenId": "{{bc-token-id}}",
 "sellingPrice": 1,
 "endorsers": {{endorsers}}
}
```

Parameters:

- `tokenId: string` – The ID of the token.
- `sellingPrice: number` – The price of the token.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "msg": "Token ID : 'artCollection1' has been posted for selling in the
marketplace"
}
```

`removeRole`
Original method name: `removeRole`

This method removes a role from a specified user. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "role": "role value (for example minter / burner)",
 "tokenDetail": "{\"tokenName\":\"tokenName value\"}",
 "endorsers": {{endorsers}}
}
```

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.

- `tokenDetail: 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"}
  ```

- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
  "msg": "Successfully removed role 'minter' from Account Id:
oaccount~60bb20c14a83f6e426e1437c479c5891e1c6477dfd7ad18b73acac5d80bc504b
(Org-Id: appdev, User-Id: user1)"
}
```

`removeTokenAdmin`
Original method name: `removeTokenAdmin`

This POST method removes a user as a `Token Admin` of the chaincode. This method can be called only by a `Token Admin` of the chaincode. An admin cannot remove themselves.

Payload:

```
{
 "orgId": "{{bc-org-id}}",
 "userId": "{{bc-user-id}}",
 "sameOrgEndorser": true
}
```

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.
- `sameOrgEndorser: boolean` – A Boolean value that indicates whether transaction endorsements must be from the same organization as the requester.

Return Value Example:

```
{
  "msg": "Successfully removed Admin (OrgId: appDev, UserId: user1)"
}
```

`safeBatchTransferFrom`
Original method name: `safeBatchTransferFrom`

This POST method transfers ownership of the specified tokens from the caller to another account. The caller of this method must be the sender of the tokens and must own the specified tokens. For fractional NFTs, if a user transfers all of their shares to another user, they lose ownership of the token.

Payload:

```
{
 "fromOrgId": "fromOrgId value",
 "fromUserId": "fromUserId value",
 "toOrgId": "toOrgId value",
 "toUserId": "toUserId value",
 "tokenIds": "[\"{{bc-token-id}}\"]",
 "quantity": "[quantity value]",
 "endorsers": {{endorsers}}
}
```

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[]` – An array of the IDs of the tokens to transfer.
- `quantity: number[]` – The list of quantities of tokens to transfer, corresponding to the token ID array.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
[
    {
        "msg": "Successfully transferred NFT token: 'FNFT' of '10' quantity
from Account-Id:
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    },
    {
        "msg": "Successfully transferred 10 FT token: 'FT' from Account-Id:
oaccount~21206f309941a2a23c4f158a0fe1b8f12bb8e2b0c9a2e1358f5efebc0c7d410e
(Org-Id: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~1089ee5122f367ee0ca38c6660298f4b81f199627e4f67f3691c0f628237974c
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    },
    {
        "msg": "Successfully transferred NFT token: 'NFT' from Account-Id:
```

```
oaccount~e88276a3be547e31b567346bdddde52d37734da4d5fae83ab2e5c98a10097371
(Org-Id: AppBldFFFFMay22, User-Id: user2) to Account-Id:
oaccount~dcee860665db8740cb77b846e823752185a1e9a185814d0acb305890f5903446
(Org-Id: AppBldFFFFMay22, User-Id: example_minter)"
    }
]
```

`suspendAccount`
Original method name: `suspendAccount`

This POST method suspends a fungible token account. It throws an error if an `accountStatus` value is not found in ledger. This method can be called only by a `Token Admin` of the chaincode.

Payload:

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "{{bc-user-id}}",
    "endorsers": {{endorsers}}
}
```

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.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "assetType": "oaccountStatus",
    "status_id":
"oaccountStatus~5a0b0d8b1c6433af9fedfe0d9e6580e7cf6b6bb62a0de6267aaf79f79d5e96
d7",
    "account_id":
"oaccount~1c568151c4acbcd1bd265c766c677145760a61c47fc8a3ba681a4cfbe287f9c1",
    "status": "suspended"
}
```

`totalNetSupply`
Original method name: `totalNetSupply`

This GET 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.

Query (Fungible Tokens):

```
/totalNetSupply?tokenDetail={"tokenId":"{{bc-token-id}}"}
```

Query (Non-Fungible Tokens):

```
/totalNetSupply?tokenDetail={"tokenName":"tokenName value"}
```

Parameters:

- `tokenDetail: 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
}
```

`totalSupply`
Original method name: `totalSupply`

This GET method returns the total number of minted tokens. This method can be called only by a `Token Admin` of the chaincode.

Query (Fungible Tokens):

```
/totalSupply?tokenDetail={"tokenId":"{{bc-token-id}}"}
```

Query (Non-Fungible Tokens):

```
/totalSupply?tokenDetail={"tokenName":"tokenName value"}
```

Parameters:

- `tokenDetail: 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
}
```

updateArtCollectionToken
Original method name: `updateArtCollectionToken`

This POST method updates token properties. Only the token owner can call this method. For NFTs, the token metadata and token URI cannot be updated after the token is minted.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc
value\",\"tokenUri\":\"tokenUri value\",\"status\":\"status value\",
\"tokenMetadata\":{\"ISIN\":\"ISIN value\",\"Segment\":\"Segment
value\",\"Issuer\":\"Issuer
value\",\"FaceValue\":999,\"IssueSize\":999,\"CouponRate\":999,\"InterestPayme
ntType\":\"InterestPaymentType
value\",\"InterestFrequency\":\"InterestFrequency
value\",\"IssueDate\":\"2023-03-28T15:16:36+00:00\",\"MaturityDate\":\"2023-03
-28T15:16:36+00:00\"},\"status\":\"status value\"}",
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to update. For more information about the properties of the token asset, see the input specification file.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
            "tokenMetadata": {
                "ISIN": "ISIN value",
                "Segment": "Segment value",
                "Issuer": "Issuer value",
                "FaceValue": 999,
                "IssueSize": 999,
                "CouponRate": 999,
                "InterestPaymentType": "simple",
                "InterestFrequency": "monthly",
                "IssueDate": "2023-03-28T15:16:36.000Z",
                "MaturityDate": "2023-03-28T15:16:36.000Z"
            },
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "bond",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy": 
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

`updateLoyaltyToken`
Original method name: `updateLoyaltyToken`

This POST method updates token properties. Only the token owner can call this method. For NFTs, the token metadata and token URI cannot be updated after the token is minted.

Payload:

```
{
 "tokenAsset": "{\"tokenId\":\"{{bc-token-id}}\",\"tokenDesc\":\"tokenDesc 
value\",\"Token_Name\":\"Token_Name value\",\"Token_to_Currency_Ratio\":999}",
 "sameOrgEndorser": true
}
```

Parameters:

- `tokenAsset: <Token Class>` – The token asset to update. For more information about the properties of the token asset, see the input specification file.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
            "assetType": "otoken",
            "events": false,
            "tokenId": "token2",
            "tokenName": "loyalty",
            "tokenDesc": "tokenDesc value",
            "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": 0
            },
            "quantity": 10,
            "createdBy":
"oaccount~85dfd98d1b99e5b8891e0a0fdcd7d2e07fc5d37958f5d2a5796290b6a9204a43",
            "creationDate": "2024-12-03T12:07:24.000Z",
            "divisible": {
                "decimal": 0
            },
            "isBurned": false,
            "isLocked": false,
            "tokenUri": "tokenUri value",
            "status": "created"
}
```

URI

Original method name: `URI`

This GET method returns the URI of a specified token. Only the token creator can call this method.

Query

```
/URI?tokenId={{bc-token-id}}
```

Parameters:

- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "tokenUri": "example.com"
}
```

Deploy and Test Generic Token Framework Chaincode

Deploying the Fungible Token Framework Chaincode

You can deploy the chaincode directly from the Oracle Blockchain Platform console or by using Blockchain App Builder. Before you deploy the chaincode, create enrollment IDs for each token user and then map the token users to their respective enrollment IDs. Specify only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to a REST Proxy.

When you deploy the token chaincode, you must call the `init` method and pass the organization ID and user ID of the `Token Admin` user.

For information about deploying from the Oracle Blockchain Platform console, see Use Advanced Deployment.

To deploy using Blockchain App Builder, complete the following steps.

1. Extract the `DepositToken.zip` archive file.
2. Import the `DepositToken` chaincode to the Blockchain App Builder extension in Visual Studio Code.
3. Edit the `.ochain.json` file to update the value of the `configFileLocation` key to the path of the `DepositToken.yml` specification file.
4. Open a terminal window and navigate to the chaincode folder, and then run the following command.

   ```
   npm install
   ```

For more information about deploying using Blockchain App Builder, see Deploy Your Chaincode Using Visual Studio Code.

Sample Process Flow for the DepositTokens Sample

A typical process flow using the fungible token framework methods follows these basic steps.

1. Admins use the `initializeDepositToken` method to initialize the deposit token system.
2. Admins use the `createAccount` and `associateTokenToAccount` methods to create accounts and associate the token to accounts for all users.
3. Admins use the `addRole` method to assign the minter role to the creator and the notary role to the approver.
4. The token creator uses the `requestMint` method to submit a request to mint deposit tokens.
5. The approver uses the `approveMint` method to review and approve the request to mint deposit tokens. The deposit tokens are credited to the creator's account.

6. The issuer uses the `getAccountBalance` method to verify that the credited amount is accurate.
7. The creator uses the `holdTokens` method to request transfer of the tokens to the issuer.
8. The approver uses the `executeHoldTokens` method to validate and approve the transfer request. The deposit tokens are transferred to the issuer's account.
9. The issuer uses the `transferTokens` method to send tokens to the first user. The deposit tokens are credited to the first user's account.
10. The first user uses the `getAccountBalance` method to verify their account balance.
11. The issuer uses the `transferTokens` method to send tokens to the first user. The deposit tokens are credited to the second user's account.
12. The second user uses the `getAccountBalance` method to verify their account balance.
13. The second user uses the `burnTokens` method to redeem their deposit tokens.

Deploying the Non-Fungible Token Framework Chaincode

You can deploy the chaincode directly from the Oracle Blockchain Platform console or by using Blockchain App Builder. Before you deploy the chaincode, create enrollment IDs for each token user and then map the token users to their respective enrollment IDs. Specify only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to a REST Proxy.

When you deploy the token chaincode, you must call the `init` method and pass the organization ID and user ID of the `Token Admin` user.

For information about deploying from the Oracle Blockchain Platform console, see Use Advanced Deployment.

To deploy using Blockchain App Builder, complete the following steps.

1. Extract the `NFTCollectiblesWithERC721.zip` archive file.
2. Import the `NFTCollectiblesWithERC721` chaincode to the Blockchain App Builder extension in Visual Studio Code.
3. Edit the `.ochain.json` file to update the value of the `configFileLocation` key to the path of the `NFTCollectiblesWithERC721-TypeScript.yml` specification file.
4. Open a terminal window and navigate to the chaincode folder, and then run the following command.

   ```
   npm install
   ```

For more information about deploying using Blockchain App Builder, see Deploy Your Chaincode Using Visual Studio Code.

Sample Process Flow for the NFTCollectiblesWithERC721 Sample

A typical process flow using the non-fungible token framework chaincode follows these basic steps.

1. Admins use the `createAccount` method to create accounts for all stakeholders, including museums/curators, buyers, and sellers.
2. Admins use the `addRole` method to assign the minter role to the curator, enabling them to mint NFTs.
3. Curators use the `createArtCollectionToken` method to mint an art collection NFT.

4. Curators use the `post` method to set the price for an NFT and post it for sale in the marketplace.
5. Buyers use the `buy` method to buy the NFT by using direct payment via a payment gateway. The purchased NFT is transferred to the buyer's account and is no longer for sale.
6. Optionally, buyers can use the `post` method to set a new price for an NFT and post it for resale in the marketplace.
7. Optionally, buyers can use the `burn` method to redeem the NFT or permanently remove it from circulation.

Deploying the Combined Token Framework Chaincode

You can deploy the chaincode directly from the Oracle Blockchain Platform console or by using Blockchain App Builder. Before you deploy the chaincode, create enrollment IDs for each token user and then map the token users to their respective enrollment IDs. Specify only one user for each enrollment. For more information about adding enrollments, see Add Enrollments to a REST Proxy.

When you deploy the token chaincode, you must call the `init` method and pass the organization ID and user ID of the `Token Admin` user.

For information about deploying from the Oracle Blockchain Platform console, see Use Advanced Deployment.

To deploy using Blockchain App Builder, complete the following steps.

1. Extract the `NFTCollectiblesWithERC1155.zip` archive file.
2. Import the `NFTCollectiblesWithERC1155` chaincode to the Blockchain App Builder extension in Visual Studio Code.
3. Edit the `.ochain.json` file to update the value of the `configFileLocation` key to the path of the `NFTCollectiblesWithERC1155-TypeScript.yml` specification file.
4. Open a terminal window and navigate to the chaincode folder, and then run the following command.

   ```
   npm install
   ```

For more information about deploying using Blockchain App Builder, see Deploy Your Chaincode Using Visual Studio Code.

Sample Process Flow for the NFTCollectiblesWithERC1155 Sample

A typical process flow using the non-fungible token framework chaincode follows these basic steps.

1. Admins use the `createAccount` method to create fungible and non-fungible token accounts for all stakeholders, including museums/curators, buyers, and sellers.
2. Admins use the `addRole` method to assign the minter role to the curator, enabling them to mint NFTs.
3. Curators use the `mintBatch` method to mint art collection NFTs.
4. Curators use the `post` method to set the price for an NFT and post it for sale in the marketplace.
5. Buyers and sellers use the `createTokenAccount` method to create consumer accounts for fungible and non-fungible tokens on the platform.

6. Buyers use the `buyWithEthCoin` method to buy the NFT with Ethereum. Buyers receive loyalty tokens from the curator during the transaction. Buyer can also pay directly via a payment gateway. The purchased NFT is transferred to the buyer's account and is no longer for sale.
7. Optionally, buyers can use the `post` method to set a new price for an NFT and post it for resale in the marketplace.
8. Optionally, buyers can use the `burnBatch` method to redeem the NFT or permanently remove it from circulation.

Deploy and Test Wrapper APIs for Generic Token Frameworks

For information about using the Postman collections included with each package, see Wrapper API Package Components.

Deploying the Fungible Token Framework Wrapper API Package

Before you can deploy the wrapper API package, you must update the required configuration variables. Some configuration variables have default values, but you must manually update any variable that contains a placeholder as its default value. Configuration variables are stored in the `terraform.tfvars` file in the wrapper API archive. For more information about deploying wrapper APIs and about configuration variables, see Wrapper APIs. The following table lists the configuration variables and their defaults values for the non-fungible token framework wrapper API package. If the default value contains placeholders, it indicates that the user must manually provide the necessary values.

Variable name	Default value	Description
`compartment_ocid`	`<compartment_ocid>`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	`<compartment_name>`	The name of the OCI compartment.
`identity_domain`	`<identity_domain>`	The identity domain to use.
`blockchain_channel`	`<blockchain_channel>`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	`<blockchain_url>`	The Oracle Blockchain Platform URL associated with the chaincode deployment.
`blockchain_chaincode`	`DepositToken`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	`true`	The sync value to include in the payload for API calls.
`blockchain_timeout`	`6000`	The timeout value to include in the payload for API calls.
`vcn_display_name`	`DepositToken`	The display name of the OCI virtual cloud network.
`application_display_name`	`DepositToken`	The display name of the OCI application.
`gateway_display_name`	`DepositToken`	The display name of API Gateway.
`deployment_display_name`	`DepositToken`	The display name of the deployment in API Gateway.

Variable name	Default value	Description
`deployment_path_prefix`	`/DepositToken`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	`deposittoken`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.
`policy_name`	`DepositToken`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

Fungible Token Framework Sample Process Flow

A typical process flow using the fungible token framework wrapper APIs follows these basic steps.

1. Admins use the `initializeDepositToken` API to initialize the deposit token system.
2. Admins use the `createAccount` and `associateTokenToAccount` APIs to create accounts and associate the token to accounts for all users.
3. Admins use the `addRole` API to assign the minter role to the creator and the notary role to the approver.
4. The token creator uses the `requestMint` API to submit a request to mint deposit tokens.
5. The approver uses the `approveMint` API to review and approve the request to mint deposit tokens. The deposit tokens are credited to the creator's account.
6. The issuer uses the `getAccountBalance` API to verify that the credited amount is accurate.
7. The creator uses the `holdTokens` API to request transfer of the tokens to the issuer.
8. The approver uses the `executeHoldTokens` API to validate and approve the transfer request. The deposit tokens are transferred to the issuer's account.
9. The issuer uses the `transferTokens` API to a user. The deposit tokens are credit to the first user's account.
10. The first user uses the `getAccountBalance` API to verify their account balance.
11. The issuer uses the `transferTokens` API to a user. The deposit tokens are credit to the second user's account.
12. The second user uses the `getAccountBalance` API to verify their account balance.
13. The second user uses the `burnTokens` API to redeem their deposit tokens.

Deploying the Non-Fungible Token Wrapper API Package

Before you can deploy the wrapper API package, you must update the required configuration variables. Some configuration variables have default values, but you must manually update any variable that contains a placeholder as its default value. Configuration variables are stored in the `terraform.tfvars` file in the wrapper API archive. For more information about deploying wrapper APIs and about configuration variables, see Wrapper APIs. The following table lists the configuration variables and their defaults values for the non-fungible token framework wrapper API package. If the default value contains placeholders, it indicates that the user must manually provide the necessary values.

Variable name	Default value	Description
`compartment_ocid`	`<compartment_ocid>`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	`<compartment_name>`	The name of the OCI compartment.
`identity_domain`	`<identity_domain>`	The identity domain to use.
`blockchain_channel`	`<blockchain_channel>`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	`<blockchain_url>`	The Oracle Blockchain Platform URL associated with the chaincode deployment.
`blockchain_chaincode`	`NFTCollectiblesWithERC721`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	`true`	The sync value to include in the payload for API calls.
`blockchain_timeout`	`6000`	The timeout value to include in the payload for API calls.
`vcn_display_name`	`NFTCollectiblesWithERC721`	The display name of the OCI virtual cloud network.
`application_display_name`	`NFTCollectiblesWithERC721`	The display name of the OCI application.
`gateway_display_name`	`NFTCollectiblesWithERC721`	The display name of API Gateway.
`deployment_display_name`	`NFTCollectiblesWithERC721`	The display name of the deployment in API Gateway.
`deployment_path_prefix`	`/NFTCollectiblesWithERC721`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	`nftcollectibleswitherc721`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.
`policy_name`	`NFTCollectiblesWithERC721`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

Non-Fungible Token Framework Sample Process Flow

A typical process flow using the non-fungible token framework wrapper APIs follows these basic steps.

1. Admins use the `createAccount` API to create accounts for all stakeholders, including museums/curators, buyers, and sellers.

2. Admins use the `addRole` API to assign the minter role to the curator, enabling them to mint NFTs.
3. Curators use the `createArtCollectionToken` API to mint an art collection NFT.
4. Curators use the `post` API to set the price for an NFT and post it for sale in the marketplace.
5. Buyers use the `buy` API to buy the NFT by using direct payment via a payment gateway. The purchased NFT is transferred to the buyer's account and is no longer for sale.
6. Optionally, buyers can use the `post` API to set a new price for an NFT and post it for resale in the marketplace.

Deploying the Combined Token Wrapper API Package

Before you can deploy the wrapper API package, you must update the required configuration variables. Some configuration variables have default values, but you must manually update any variable that contains a placeholder as its default value. Configuration variables are stored in the `terraform.tfvars` file in the wrapper API archive. For more information about deploying wrapper APIs and about configuration variables, see Wrapper APIs. The following table lists the configuration variables and their defaults values for the combined token framework wrapper API package. If the default value contains placeholders, it indicates that the user must manually provide the necessary values.

Variable name	Default value	Description
`compartment_ocid`	`<compartment_ocid>`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	`<compartment_name>`	The name of the OCI compartment.
`identity_domain`	`<identity_domain>`	The identity domain to use.
`blockchain_channel`	`<blockchain_channel>`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	`<blockchain_url>`	The Oracle Blockchain Platform URL associated with the chaincode deployment.
`blockchain_chaincode`	`WholesaleCBDC`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	`true`	The sync value to include in the payload for API calls.
`blockchain_timeout`	`6000`	The timeout value to include in the payload for API calls.
`vcn_display_name`	`NFTCollectiblesWithERC1155`	The display name of the OCI virtual cloud network.
`application_display_name`	`NFTCollectiblesWithERC1155`	The display name of the OCI application.
`gateway_display_name`	`NFTCollectiblesWithERC1155`	The display name of API Gateway.
`deployment_display_name`	`NFTCollectiblesWithERC1155`	The display name of the deployment in API Gateway.

Variable name	Default value	Description
`deployment_path_prefix`	`/NFTCollectiblesWithERC1155`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	`nftcollectibleswitherc1155`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.
`policy_name`	`NFTCollectiblesWithERC1155`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

Combined Token Framework Sample Process Flow

A typical process flow using the combined token framework wrapper APIs follows these basic steps.

1. Admins use the `createAccount` API to create fungible and non-fungible token accounts for all stakeholders, including museums/curators, buyers, and sellers.
2. Admins use the `addRole` API to assign the minter role to the curator, enabling them to mint NFTs.
3. Curators use the `mintBatch` API to mint art collection NFTs.
4. Curators use the `post` API to set the price for an NFT and post it for sale in the marketplace.
5. Buyers and sellers use the `createTokenAccount` API to create consumer accounts for fungible and non-fungible tokens on the platform.
6. Buyers use the `buyWithEthCoin` API to buy the NFT with Ethereum. The purchased NFT is transferred to the buyer's account and is no longer for sale.
7. Optionally, buyers can use the `post` API to set a new price for an NFT and post it for resale in the marketplace.
8. Optionally, buyers can use the `burnBatch` API to redeem the NFT or permanently remove it from circulation.

6

Blockchain App Builder Enhancements

Oracle Blockchain Platform Digital Assets Edition includes an enhanced version of Blockchain App Builder.

For information about the standard version of Blockchain App Builder, see Build Chaincodes with Low-Code Blockchain App Builder.

The following functions are supported by the enhanced version of Blockchain App Builder.

- Automatic generation of complete chaincode applications for wholesale CBDC and bond marketplace scenarios.
- Automatic generation of wrapper API packages for API Gateway, with dedicated endpoints for each chaincode method. You can configure the names of the endpoints to match your organization's naming conventions.
- Support for chaincode events, which can send enable real-time notifications and trigger workflows.
- Support for endorsement parameters in generated Postman collections.
- New account, role, and transaction functions in the extended Token Taxonomy Framework standard.

Note:

Blockchain App Builder prerequisites include Go v1.23.2 or later, but not Go v1.24 or later. For more information, see Install and Configure Blockchain App Builder CLI. and Install and Configure the Blockchain App Builder Extension for Visual Studio Code.

Wrapper APIs

The enhanced version of Blockchain App Builder can generate wrapper APIs for API Gateway, which support dedicated endpoints for each chaincode method.

A wrapper API is an abstraction layer over an Oracle Blockchain Platform endpoint. Wrapper APIs support configuring common parameters and API-specific endpoint names, which can simplify usage and clarity.

In the base version of Oracle Blockchain Platform, an API endpoint requires parameters including the chaincode name, timeout and sync values, arguments (including the method name), channel name, and instance URL. These parameters must be passed with every invocation. Because most of these parameters are common to all methods, you can use the wrapper API functionality to create an endpoint for each method, which you then invoke using only the method parameters. Instead of a common endpoint, either a transaction or a query, you can have a custom endpoint name for each method. Whereas API endpoints in Oracle Blockchain Platform use POST requests, wrapper APIs support both POST and GET requests. You can also add an extra layer of authentication with wrapper APIs by using the API Gateway authentication mechanism.

The following example shows the createAccount method API in Oracle Blockchain Platform.

Endpoint: `https://blockchain.example.com:7443/restproxy/api/v2/channels/default/transactions`

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "createAccount",
        "{{bc-org-id}}",
        "{{bc-user-id}}",
        "fungible",
        "{\"max_daily_amount\":10000,\"max_daily_transactions\":100}"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}}
}
```

If you configure wrapper APIs, you can make the same method call as shown in the following example.

Endpoint: `https://apigateway.example.com/appbuilder/createAccount`

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "user1",
    "tokenType": "fungible",
    "dailyLimits":
"{\"max_daily_amount\":10000,\"max_daily_transactions\":100}",
}
```

Generate Wrapper APIs Using the CLI

The enhanced version of Blockchain App Builder includes the `generateFnAPI` command, which generates wrapper APIs for all controller APIs in a chaincode project.

```
Usage: ochain generateFnAPI [options]
Generation of WrapperAPI package for controller functions
Options:
-h, --help                  Output command usage information
-D, --debug                 Enable debug logging
-m, --mapping <mapping>     Info about functions endpoints mapping is
mandatory.
-c, --config <config>       Info about configuration variables is mandatory.
-a, --all <all>             Generate the wrapperAPI package for entire
controller functions
-p, --project <path>        Path to the Ochain chaincode project to generate
the WrapperAPI package. If not specified, it defaults to current directory.
```

The following example shows how to generate wrapper APIs for a local chaincode project.

```
ochain generateFnAPI --mapping <info object about function mapping> --config
<info about the config variables> --project <Path to the Ochain chaincode
project> --out <Path to the generated postman collection>
```

When you run the `generateFnAPI` command, you are prompted whether to overwrite any previous wrapper API packages that exist in the selection location.

API Gateway limits the number of routes that can be created in a single deployment to 50. If you try to use the `generateFnAPI` command to generate more than 50 wrapper APIs, you are prompted whether to create multiple gateways. Creating multiple gateways with the same deployment path prefix maintains a consistent base path across all wrapper API endpoints. If you choose not to create multiple gateways, wrapper API generation is canceled.

Command Options

-m, --mapping <mapping>
The mapping option defines the configuration of controller APIs, specifying the endpoint names and which controller APIs require wrapper API generation. The map is a JSON object of key/value pairs, where the key is the name of the controller API and the value is the custom endpoint to associate with that controller API.
The following example shows a mapping for chaincode that uses the extended Token Taxonomy Framework standard.

```
ochain generateFnAPI --mapping '{"addConversionRate":"/
addConversionRateTest", "addTokenAdmin": "/addTokenAdminTest", "approveBurn":
"/approveBurnTest", "createAccount": "/createAccount",
"associateTokenToAccount": "/associateTokenToAccountTest",
"getAllOrgAccounts": "/getAllOrgAccounts"}' --config <info about the config
variables> --project <Path to the Ochain chaincode project> --out <Path to
the generated postman collection>
```

-c, --config <config>
The config option is used to pass the configuration variables that are required for the wrapper API package. It is a JSON object of key/value pairs, where the key is the configuration variable name and the value is the configuration variable value. All of the configuration variables are mandatory for generating a wrapper API package. If any of the following variables are not passed to the command, wrapper API generation fails. You can edit these values after you create the wrapper API by extracting the wrapper API `.zip` file.

Variable name	Description
`compartment_ocid`	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).
`compartment_name`	The name of the OCI compartment.
`identity_domain`	The identity domain to use.
`blockchain_channel`	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.
`blockchain_url`	The Oracle Blockchain Platform URL associated with the chaincode deployment.

Variable name	Description
`blockchain_chaincode`	The name of the chaincode to generate wrapper APIs for.
`blockchain_sync`	The sync value to include in the payload for API calls.
`blockchain_timeout`	The timeout value to include in the payload for API calls.
`vcn_display_name`	The display name of the OCI virtual cloud network.
`application_display_name`	The display name of the OCI application.
`gateway_display_name`	The display name of API Gateway.
`deployment_display_name`	The display name of the deployment in API Gateway.
`deployment_path_prefix`	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The `deployment_path_prefix` variable must begin with a slash (/).
`ocir_repo_name`	The OCI Registry repository name. The `ocir_repo_name` variable must be all lowercase letters.
`policy_name`	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization

The following example shows a mapping for chaincode that uses the extended Token Taxonomy Framework standard.

```
ochain generateFnAPI -m '{"addConversionRate":"/addConversionRateTest",
"addTokenAdmin": "/addTokenAdminTest", "approveBurn": "/approveBurnTest",
"createAccount": "/createAccount", "associateTokenToAccount": "/
associateTokenToAccountTest", "getAllOrgAccounts": "/getAllOrgAccounts"}' -c
'{"compartment_ocid": "compartment_ocid value", "compartment_name":
"compartment_name value", "identity_domain" : "OracleIdentityCloudService",
"blockchain_channel": "default", "blockchain_url": "blockchain_url value",
"blockchain_chaincode": "blockchain_chaincode value", "blockchain_sync":
true, "blockchain_timeout": 6000, "vcn_display_name": "vcn_display_name
value", "application_display_name": "application_display_name value",
"gateway_display_name": "gateway_display_name value",
"deployment_display_name": "deployment_display_name value",
"deployment_path_prefix": "deployment_path_prefix value", "ocir_repo_name":
"ocir_repo_name value", "policy_name": "policy_name value"}'
```

-a, --all <all>
The all option generates wrapper API for all controller APIs that are defined in the chaincode. If `true`, this option overrides the mapping option. Every wrapper API is assigned a default endpoint, which corresponds to the API name. If you specify the all option, you must still provide an empty JSON object ({}) for the mapping option.

The following example shows a mapping for chaincode that uses the extended Token Taxonomy Framework standard.

```
ochain generateFnAPI -m '{}' -c '{"compartment_ocid": "compartment_ocid
value", "compartment_name": "compartment_name value", "identity_domain" :
"OracleIdentityCloudService", "blockchain_channel": "default",
"blockchain_url": "blockchain_url value", "blockchain_chaincode":
"blockchain_chaincode value", "blockchain_sync": true, "blockchain_timeout":
6000, "vcn_display_name": "vcn_display_name value",
"application_display_name": "application_display_name value",
"gateway_display_name": "gateway_display_name value",
"deployment_display_name": "deployment_display_name value",
"deployment_path_prefix": "deployment_path_prefix value", "ocir_repo_name":
"ocir_repo_name value", "policy_name": "policy_name value"}' -a true
```

Generate Wrapper APIs Using Visual Studio Code

The enhanced version of Blockchain App Builder enables you to generate wrapper APIs in Visual Studio Code.

When you generate wrapper APIs in Visual Studio Code, you are prompted whether to overwrite any previous wrapper API packages that exist in the selection location.

API Gateway limits the number of routes that can be created in a single deployment to 50. If you try to generate more than 50 wrapper APIs, you are prompted whether to create multiple gateways. Creating multiple gateways with the same deployment path prefix maintains a consistent base path across all wrapper API endpoints. If you choose not to create multiple gateways, wrapper API generation is canceled.

1. Select the chaincode project in the Chaincodes pane.
2. Right-click the chaincode name and then select **Generate Wrapper API Package**. The Oracle Blockchain Platform - Wrapper API Package Generation pane opens.
3. Enter the configuration variables as described in the following table.

Field	Description	Default value
Compartment OCID	The OCID of the compartment in Oracle Cloud Infrastructure (OCI).	none
Compartment Name	The name of the OCI compartment.	none
Identity Domain	The identity domain to use.	none
Blockchain Channel	The name of the Oracle Blockchain Platform channel where the chaincode is deployed.	none
Blockchain URL	The Oracle Blockchain Platform URL associated with the chaincode deployment.	none
Location	The system directory to generate the wrapper API package in.	none

Field	Description	Default value
Deployment Path Prefix	The deployment path prefix in API Gateway, which specifies the path where routes are deployed. The deployment path prefix must begin with a slash (/).	`/<ChaincodeName>`
Blockchain Sync	The sync value to include in the payload for API calls.	`true`
Blockchain Timeout	The timeout value to include in the payload for API calls.	`6000`
Policy Name	The name of the policy that enables controlled management and access to APIs through defined permissions for groups and compartments within the organization	`ChaincodeName`
VCN Display Name	The display name of the OCI virtual cloud network.	`ChaincodeName`
Application Display Name	The display name of the OCI application.	`ChaincodeName`
Gateway Display Name	The display name of API Gateway.	`ChaincodeName`
Deployment Display Name	The display name of API Gateway.	`ChaincodeName`
OCI Repo Name	The OCI Registry repository name, which must be all lowercase letters.	`ChaincodeName` in lowercase characters

4. Click **Save**.
5. Click the **Mapping** tab. The Mapping pane shows a table of all of the controller APIs that are available in the chaincode. Each method has a corresponding **API Endpoint**, which you can edit as needed. Select the corresponding **Generate Wrapper APIs** check box to include the controller API in the wrapper API package. Click **Select All** to include all controller APIs.
6. Click **Save**.
7. Click **Generate**. The wrapper API package and associated Postman collection are generated.

Wrapper API Package Components

Wrapper API packages contain an archive file of the wrapper APIs, a Terraform script for deployment, and a corresponding Postman collection.

- The wrapper API archive file is named `<ChaincodeName>WrapperAPI.zip`. It also contains a Terraform script that must be deployed to the stack resource.
- The Postman collection file is named `<ChaincodeName>_WrapperAPI.postman_collection.json`. You can use this collection to invoke all of the wrapper APIs.

Wrapper API Package

The wrapper API package contains a Terraform script that provisions all of the Oracle Cloud Infrastructure (OCI) resources that are necessary for creating the wrapper APIs. There are additional Terraform support files and an Oracle Functions folder for each API. The wrapper API packages creates the following OCI resources.

- Virtual Cloud Network (VCN): Establishes the network infrastructure for communication.
- Applications (Oracle Functions): Deploys serverless functions to handle API logic.
- API Gateway: Creates the gateway to manage and route API requests.
- API Deployment: Configures and deploys the APIs on the API Gateway.
- API Deployment Policy: Sets up the necessary IAM policies to enable secure access.
- OCI Registry: Provides a container registry for managing Docker images.

After you generate wrapper APIs, if you want to change any configuration variables, you can update them in Visual Studio Code, or you can extract the wrapper API package and update the `terraform.tfvars` file with updated endpoints and resource names. Edit the `function_paths` variable the `terraform.tfvars` file to update an endpoint. The `function_paths` variable is a JSON object where the key is the API name and the value is another JSON object with the following two keys:

- `path`: Defines the endpoint for the API
- `type`: Specifies the request type (POST or GET)

The following text shows an example of a `function_paths` variable.

```
function_paths="{\"activateAccount\":{\"path\":\"/activateAccount\",\"type\":
[\"POST\"]}}"
```

Postman Collection

The Postman collection includes updated endpoints and payloads for all APIs. The following code shows an example payload.

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "user1",
    "tokenType": "fungible",
    "applicationGroups": "[\"application_groups value\"]",
    "dailyLimits":
"{\"max_daily_amount\":10000,\"max_daily_transactions\":100}",
    "endorsers": {{endorsers}}
}
```

The following table shows the Postman collection variables.

Variable	Description	Default value
`bc-admin-user`	The admin user, which has admin role where it has access to all POST requests. By default, this user is the caller of all POST requests in the chaincode.	`bc-admin-user value`
`bc-admin-user-password`	Admin user password.	`bc-admin-user-password value`
`bc-org-id`	The default organization ID in all POST requests where `orgId` is the parameter name	`bc-org-id value`

Variable	Description	Default value
`bc-user-id`	The default user ID in all POST requests where `userId` is the parameter name	`bc-user-id value`
`bc-token-id`	The default token ID in all POST requests where `tokenId` is the parameter name	`bc-token-id value`
`endorsers`	The endorsers array lists the specific peers (for example: peer1, peer2) to endorse this transaction.	`["org1-xyz-abc.blockchain.ocp.oraclecloud.com:20009", "org2-xyz-abc.blockchain.ocp.oraclecloud.com:20009"]`
`api-gateway-endpoint`	The endpoint of each request, which serves as a base path for wrapper API endpoints. If there are fewer than 50 APIs, a single endpoint is used. If there are more than 50 APIs, the endpoints are dynamically generated as api-gateway-endpoint1, api-gateway-endpoint2, and so on, based on the number of APIs.	`https://xyz.apigateway.region.oci.customer-oci.com/CBDC`
`peer`	This variable exists only for the confidential chaincode wrapper API Postman collection, which requires the peer header for all setter APIs.	`org-xyz-abc.blockchain.ocp.oraclecloud.com:20009`

After you deploy the wrapper API package, the output from the stack resource deployment is a JSON object that contains the gateway endpoint values. If you generate more than 50 APIs, multiple gateway endpoints are generated, one for every 50 APIs. You must update the Postman collection variables related to these endpoints. The endpoint-related variables in the Postman collection must be updated with the appropriate values from the output of the wrapper API package deployment in the Stack Resource Manager.

All setter APIs in the wrapper API Postman collection include either the `endorsers` or `sameOrgEndorser` parameter in the request payload. The information that specifies which APIs require the `sameOrgEndorser` parameter is defined in the `sameOrgEndorserOptionInWrapperAPI` parameter in the `.ochain.json` file in the chaincode. APIs listed in this parameter will have `sameOrgEndorser` set to true in their payloads. All other setter APIs will include the `endorsers` parameter instead. The following example shows the parameter for the wholesale CBDC chaincode.

```
"sameOrgEndorserOptionInWrapperAPI":
["addConversionRate","addTokenAdmin","addTokenAuditor","approveBurn","approveM
int","burnTokens","createExchangePoolAccounts","deleteHistoricalTransactions",
"initializeCBDCToken","initializeExchangePoolUser","mintWithFundingExchangePoo
l","rejectBurn","rejectMint","removeTokenAdmin","removeTokenAuditor","requestB
urn","requestMint","updateCBDCToken","updateConversionRate"]
```

You can customize the `sameOrgEndorserOptionInWrapperAPI` parameter in the `.ochain.json` file as needed. When you generate wrapper APIs, the specified APIs will then include the `sameOrgEndorser` parameter as `true` in their payloads.

For more details about using Postman collections, see the following topics.

- Generate a Postman Collection Using the CLI
- Generate a Postman Collection Using Visual Studio Code
- Endorsement Support in Postman Collections

Deploy Wrapper APIs

Oracle Blockchain Platform Digital Assets Edition allows the generation of wrapper APIs for all controller APIs within App Builder generated chaincode. This topic provides detailed steps for deploying the wrapper APIs package in the stack resource manager and testing the wrapper APIs using the associated Postman collection.

When the wrapper API package is generated using Blockchain App Builder, it creates two files with default naming conventions in the chosen directory:

1. `<`*`ChaincodeName`*`>WrapperAPI.zip`: This file contains the Terraform script required for deploying the wrapper APIs in Oracle Cloud Infrastructure Resource Manager.
2. `<`*`ChaincodeName`*`>_WrapperAPI.postman_collection.json`: A Postman collection used for testing the wrapper APIs after deployment.

To deploy the wrapper APIs, the `WrapperAPI.zip` file must be used within OCI Resource Manager.

Prerequisites
The prerequisites for deploying the wrapper APIs package are as follows:

- The wrapper APIs package must be generated using App Builder.
- The package should include all required configuration variable values and endpoint values for the wrapper APIs.

Deploying the Wrapper API package

1. Log in to OCI.

   Open the OCI console and sign in: https://cloud.oracle.com/.

   Ensure you're in the compartment where the stack will be deployed.
2. Expand the **Navigation** menu. Select **Developer Services**. Under **Resource Manager**, select **Stacks**.

   The Resource Manager interface for stacks opens. You can view existing stacks, create new stacks, and manage your infrastructure as code using Terraform configurations.
3. Create a new stack by clicking **Create Stack**.
4. Configure the stack information.
   a. In the **Terraform Configuration** section, select **My Configuration**.
   b. In **Stack Configuration**, **Terraform configuration source**, select **.Zip file**.
   c. Click **Browse** and navigate to your wrapper APIs Zip file.
   d. Once it's uploaded, ensure the stack name is correct.
   e. All remaining settings can be kept as default. Click **Next**.

5. Configure the stack variables.
   a. Verify that all configuration variable values are accurate and align with the configuration variables specified in the wrapper API package to meet the requirements for your deployment.
   b. For **ocir_user_name**, enter the email address associated with your OCI account.
   c. For **ocir_user_password** enter your password, which is your Oracle Cloud Infrastructure Auth Token. This token can be generated in the OCI console under your user settings. The Auth Token serves as a password for logging into the Oracle Cloud Infrastructure Registry (OCIR).

      For detailed instructions on generating the Auth Token, refer to Generating an Auth Token.
   d. All remaining settings can be kept as default. Click **Next**.
6. Review the stack information. If everything is correct, click **Create**.
7. Execute the Terraform plan.
   a. On the stack's **Details** page, click **Plan** to initiate the creation of an execution plan.
   b. Optionally configure the job plan. You can edit the default name or select advanced options to change provider version of adjust settings.
   c. Click **Plan** to create and run the plan job.

      This will parse your Terraform configuration, generate an execution plan, and execute the plan which does the following:
      - Validates the Terraform script to ensure there are no syntax or configuration errors.
      - Simulates the resource creation process without making any changes to the actual infrastructure.
      - Provides an output summary that lists the resources the script intends to create, modify, or destroy.
   d. Monitor the status in the Jobs section of your stack. Once the plan completes successfully, review the output to confirm the number of resources that will be created and verify that there are no issues.
8. Apply the Terraform plan to the stack.
   a. On the stack's **Details** page, click **Apply**.
   b. In the Terraform Apply panel, click **Apply**.

      This will do the following:
      - Provisions all of the following resources as defined in the Terraform script:
        – Virtual cloud network (VCN)
        – Applications (OCI Functions)
        – API Gateway
        – API deployment
        – IAM Policy
        – OCI Registry repository
      - Creates all the required infrastructure for the Wrapper APIs.
   c. Monitor the status in the Jobs section of your stack to ensure the job completed successfully with no errors.

9. Retrieve the API Gateway endpoints.
    a. Once the Apply job has finished, go to the **Outputs** section of the Apply job page to view the generated output values.
    b. The output will return a JSON object where the keys represent endpoint names and the values are their corresponding endpoint values.

       For example:

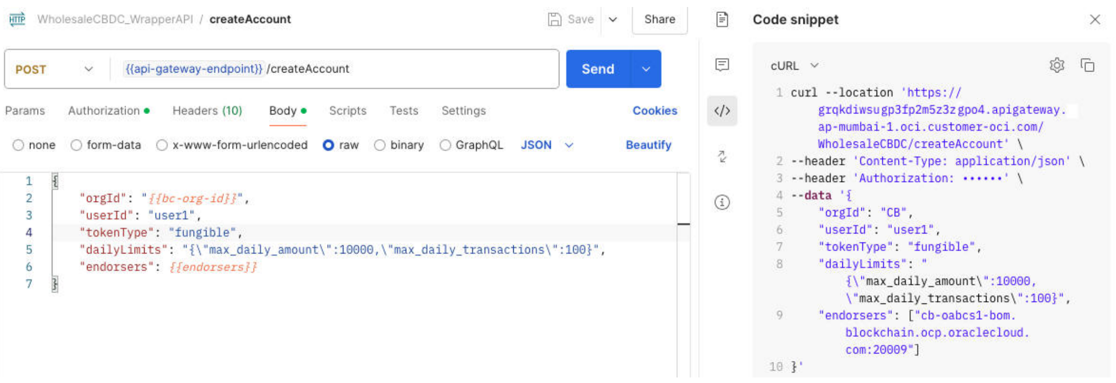
       ```
       api_gateway_endpoints{"api_gateway_endpoint":"https://
       grqkdiwsugp3fp2m5z3zgpo4.apigateway.ap-mumbai-1.oci.customer-oci.com/
       WholesaleCBDC"}
       ```

    c. Copy the endpoint - it's needed to updating the API Postman collection.

Testing Wrapper APIs with Postman
You can test the APIs in Postman. Configure all the variables in the wrapper API Postman collection by assigning the appropriate values. The `api_gateway_endpoint` value must be obtained from the output of the stack deployment in OCI, where the dynamically generated gateway endpoints will be provided.

Once the variables in the Postman collection have been updated, you can test the wrapper APIs. Each API request in the collection includes the necessary payload specific to the wrapper API.

Figure 6-1 Testing an API Wrapper Collection with Postman



POST {{api-gateway-endpoint}} /createAccount

```
{
    "orgId": "{{bc-org-id}}",
    "userId": "user1",
    "tokenType": "fungible",
    "dailyLimits": "{\"max_daily_amount\":10000,\"max_daily_transactions\":100}",
    "endorsers": {{endorsers}}
}
```

```
curl --location 'https://
    grqkdiwsugp3fp2m5z3zgpo4.apigateway.
    ap-mumbai-1.oci.customer-oci.com/
    WholesaleCBDC/createAccount' \
--header 'Content-Type: application/json' \
--header 'Authorization: ••••••' \
--data '{
    "orgId": "CB",
    "userId": "user1",
    "tokenType": "fungible",
    "dailyLimits": "
        {\"max_daily_amount\":10000,
        \"max_daily_transactions\":100}",
    "endorsers": ["cb-oabcs1-bom.
        blockchain.ocp.oraclecloud.
        com:20009"]
}'
```

Troubleshooting
The following are some commonly encountered issues and their solutions.

Unauthorized: Invalid ocir_user_name or ocir_user_password
This error occurs when either the username or password credentials are incorrect. To resolve this issue, follow these steps:

1. Navigate to the **Variables** section in the stack Details page.
2. Click **Edit Variables** to modify the username and password values.
3. Update the credentials with the correct information.
4. Once the changes are made, click **Apply** to redeploy the stack.

This should resolve the authentication issue and allow the deployment to proceed successfully.

Deployment Failure - "denied: Anonymous users are only allowed read access on public repos"
Occasionally, the deployment might fail with the following error:

```
denied: Anonymous users are only allowed read access on public repos
```

This is an intermittent issue that is related to Docker operations within the stack's backend. Specifically, this error occurs when attempting to push Docker images to the Oracle Cloud Infrastructure Registry (OCIR) without proper authentication.
While the wrapper API package includes a script that performs `docker login` before pushing the images, ensuring proper authentication, this error can still occur sporadically. It typically happens when the Docker authentication step is not properly recognized by the system at the time of image push.

If this error occurs, follow these steps to resolve the issue:

1. Destroy Existing Resources:

   Click **Destroy** in the OCI Stack Resource Manager to destroy all the resources created during the deployment process.

2. Reapply the Stack:

   After destroying the resources, click **Apply** again to redeploy the stack. This will trigger the creation of the necessary resources, including proper Docker authentication, and should resolve the issue.

Generic Deployment Failure: Intermittent Issues
Sometimes, stack deployment may fail due to random like docker related issues. In such cases, follow these steps to resolve the issue:

1. Click **Destroy** to remove all the resources created during the failed deployment.
2. Once the resources are destroyed, navigate back to the stack and click **Apply** to redeploy the stack.

This process often resolves intermittent issues, allowing the deployment to proceed successfully.

Chaincode Events

The enhanced version of Blockchain App Builder can generate chaincode events for token operations.

Chaincode events are specific notifications that are emitted when transactions run. Events include transaction information that can be used to notify external systems about specific conditions or changes in the blockchain ledger state. You can use chaincode events to enable real-time integration and interaction with applications that are not on the blockchain, and to facilitate event-driven workflows and monitoring across the blockchain environment. Chaincode events have two components, the event name and the payload.

Chaincode events are supported for all Blockchain App Builder specification files. If you enable chaincode events, all controller functions in your scaffolded project will emit events, except for getter methods. For example, in a token scenario, chaincode events will be emitted when tokens are minted, transferred, burned, or locked

You use the Boolean `events` parameter in the specification file to enable chaincode events, as shown in the following example.

```
assets:
    - name: FiatMoneyTOK # Asset name
      type: token  # Asset type
      events: true  # Generate events for create, update and delete APIs
```

If you enable events, the controller functions in your scaffolded chaincode project will include event creation methods, as shown in the following examples.

TypeScript:

```
@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 });
  await this.Ctx.Model.createEvent(EVENT_NAME.CREATE_ACCOUNT, { org_id,
user_id, token_type });
  return await this.Ctx.Account.createAccount(org_id, user_id, token_type);
}
```

Go:

```
func (t *Controller) CreateAccount(org_id string, user_id string, token_type
string, daily_limits ...account.AccountDailyLimits) (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())
    }
    err = t.Ctx.Model.CreateEvent(constants.CreateAccountEventName,
map[string]interface{}{"org_id": org_id, "user_id": user_id, "token_type":
token_type})
    if err != nil {
        return nil, err
    }
    return t.Ctx.Account.CreateAccount(org_id, user_id, token_type,
daily_limits...)
}
```

Chaincode events use the following default values for the name and payload components. You can modify the default values as needed.

EventName
The name of the controller function.

Payload
A JSON object that contains all of the input parameters of the controller function.

A new `events` parameter was also added to the token details in the extended Token Taxonomy Framework and ERC-1155 standards. If the `events` parameter in the specification file is set to true, the `events` parameter in the generated token is set to true. If the `events` parameter in the specification file is set to false or not defined, the `events` parameter in the generated token is set to false. The following examples show a token with the new `events` parameter for both TypeScript and Go.

```
{
    "metadata": {
        "paintingName": "monalisa",
        "description": "monalisa painting",
        "image": "image link",
        "painterName": "Leonardo da Vinci"
    },
    "assetType": "otoken",
    "events": true,
    "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
}
```

```
{
    "AssetType": "otoken",
    "Behavior": [
        "indivisible",
        "singleton",
        "mintable",
        "transferable",
        "burnable",
        "roles"
    ],
    "CreatedBy":
"oaccount~42e89f4c72dfde9502814876423c6da630d466e87436dd1aae201d347ad1288d",
    "CreationDate": "2022-12-29T09:57:03+05:30",
    "Events": true,
    "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"
}
```

Generating Events

The `stub.setEvent` method enables the chaincode to create and emit an event when a transaction runs. The following code shows the TypeScript version of the method.

```
async setEvent(eventName: string, payload: Buffer): Promise<void>
```

In this example, `eventName` is the name to assign to the event, and `payload` is data associated with the event. The payload can contain any information that you want to send with the event, typically serialized in JSON format.

Chaincode Events for Batch Methods

The enhanced ERC-1155 standard supports batch methods. Batch methods operate on multiple tokens that are passed as parameters. For batch methods, chaincode events are emitted for only the tokens where the `events` parameter is set to `true` in the specification file.

For each transaction completed in the batch method, a corresponding chaincode event is generated. The payload of each chaincode event contains the transaction details. For example, if you use the `BatchTransfer` method to transfer quantities of five different tokens, five corresponding chaincode events are emitted. The payload of each event contains the token details and the quantity transferred, along with common parameters that are applicable to all batch transfers.

The following example shows events code using the enhanced ERC-1155 standard.

```
@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);
    let tokenAssets = [];
    for (let i = 0; i < tokenIds.length; i++) {
      const tokenAsset = await this.Ctx.ERC1155Token.get(tokenIds[i]);
      tokenAssets.push(tokenAsset);
    }
    await this.Ctx.Model.createEventForBatch(EVENT_NAME.BATCH_TRANSFER_FROM,
{ fromOrgId, fromUserId, toOrgId, toUserId }, quantity, tokenAssets);
    return await this.Ctx.ERC1155Token.batchTransferFrom(fromAccountId,
toAccountId, tokenIds, quantity);
  }
```

Chaincode Events for Multiple Assets

The enhanced Token Taxonomy Framework and ERC-1155 standards support defining more than one token asset in a specification file. The chaincode event behavior is different depending on whether a method is token-specific (such as creating or updating a token) or common (such as minting or burning).

For token-specific methods, chaincode events are generated for only the tokens where the `events` parameter is set to `true` in the specification file.

For common methods, chaincode events are generated in the scaffolded project if any token has the `events` parameter set to `true` in the specification file. The actual chaincode event behavior is based on the number of token ID parameters passed to the method.

- If a single token ID is passed as a parameter, chaincode events are generated only if the `events` parameter in the corresponding token details is set to true.
- If multiple token IDs are passed as parameters, chaincode events are generated only if the `events` parameter in any one of the token details is set to true.
- If no token ID is passed as a parameter, chaincode events are always generated.

The following list shows common methods that must be passed two tokens. This list applies to the extended Token Taxonomy Framework standard.

- `addConversionRate(from_token_id: string, to_token_id: string, token_conversion_rate: number)`

- `updateConversionRate(from_token_id: string, to_token_id: string, token_conversion_rate: number)`
- `tokenConversion(from_token_id: string, to_token_id: string, to_org_id: string, to_user_id: string,token_quantity: number)`
- `exchangeToken(fromTokenId: string, fromOrgId: string, fromUserId: string, fromTokenQuantity: number, toTokenId: string, toOrgId: string,toUserId: string,toTokenQuantity: number)`

The following list shows common methods that do not take tokens as arguments The following list applies to the extended Token Taxonomy Framework standard.

- `addTokenAdmin(org_id: string, user_id: string)`
- `removeTokenAdmin(org_id: string, user_id: string)`
- `addOrgAdmin(org_id: string, user_id: string)`
- `removeOrgAdmin(org_id: string, user_id: string)`
- `createAccount(org_id: string, user_id: string, token_type: string)`
- `deleteHistoricalTransactions(time_to_expiration: Date)`
- `initializeExchangePoolUser(org_id: string, user_id: string)`

This list shows common methods that do not take tokens as arguments for the extended ERC-1155 standard.

- `addTokenAdmin(orgId: string, userId: string)`
- `removeTokenAdmin(orgId: string, userId: string)`
- `createAccount(orgId: string, userId: string, ftAccount: boolean, nftAccount: boolean)`
- `createUserAccount(orgId: string, userId: string)`
- `createTokenAccount(orgId: string, userId: string, tokenType: TokenType)`
- `addTokenSysRole(orgId: string, userId: string, role: string)`
- `removeTokenSysRole(orgId: string, userId: string, role: string)`
- `transferTokenSysRole(fromOrgId: string, fromUserId: string, toOrgId: string, toUserId: string, role: string)`
- `deleteHistoricalTransactions(time_to_expiration: Date)`

TypeScript SDK Methods for Chaincode Events

`createEvent`
This method generates events based on a specified event name and payload.

```
public async createEvent(eventName: any, eventPayload: any, assets?: any)
```

Parameters:

- `eventName: string` – The event name to use when generating events.
- `eventPayload: map[string]interface{}` – The event payload to use when generating events.

- `assets` – Optionally, the token asset can be passed as a parameter to the method.

`createEventForBatch`
This method generates events for batch operations such as the `mintBatch` or `burnBatch` methods.

```
public async createEventForBatch(eventName: any, eventPayload: any,
quantities: number[], assets: any)
```

Parameters:

- `eventName: string` – The event name to use when generating events.
- `eventPayload: map[string]interface{}` – The event payload to use when generating events.
- `quantities: number[]` – A list of amounts, corresponding to each token ID, that represent the number of tokens to use in the batch method transactions.
- `assets` – Optionally, the token asset can be passed as a parameter to the method.

Go SDK Methods for Chaincode Events

`CreateEvent`
This method generates events based on a specified event name and payload.

```
func (m *Model) CreateEvent(eventName string, eventPayload
map[string]interface{}, assets ...interface{})
```

Parameters:

- `eventName: string` – The event name to use when generating events.
- `eventPayload: map[string]interface{}` – The event payload to use when generating events.
- `assets` – Optionally, the token asset can be passed as a parameter to the method.

`CreateEventForBatch`
This method generates events for batch operations such as the `mintBatch` or `burnBatch` methods.

```
func (m *Model) CreateEventForBatch(eventName string, eventPayload
map[string]interface{}, quantities []float64, assets []interface{})
```

Parameters:

- `eventName: string` – The event name to use when generating events.
- `eventPayload: map[string]interface{}` – The event payload to use when generating events.
- `quantities: []float64` – A list of amounts, corresponding to each token ID, that represent the number of tokens to use in the batch method transactions.
- `assets` – Optionally, the token asset can be passed as a parameter to the method.

Event Generation in Blockchain App Builder Chaincode

If the `events` parameter in the specification file is set to true, chaincode events code is generated for all controller APIs except for those designated as getter APIs. The following examples show controller APIs with chaincode events enabled for both TypeScript and Go.

```
@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 });
  await this.Ctx.Model.createEvent(EVENT_NAME.CREATE_ACCOUNT, { org_id,
user_id, token_type });
  return await this.Ctx.Account.createAccount(org_id, user_id, token_type);
}
```

```
func (t *Controller) CreateAccount(org_id string, user_id string, token_type
string, daily_limits ...account.AccountDailyLimits) (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())
    }
    err = t.Ctx.Model.CreateEvent(constants.CreateAccountEventName,
map[string]interface{}{"org_id": org_id, "user_id": user_id, "token_type":
token_type})
    if err != nil {
        return nil, err
    }
    return t.Ctx.Account.CreateAccount(org_id, user_id, token_type,
daily_limits...)
}
```

Chaincode events are generated with the following default values. You can modify these values as needed.

- `EventName`: The name of the controller function.
- `Payload`: A JSON object that contains all of the input parameters of the controller function.

Token Taxonomy Framework Enhancements

The enhanced version of Blockchain App Builder includes new functionality related to the extended Token Taxonomy Framework standard.

Daily Transaction Limits

You can restrict the number of transactions an account can complete daily, as well as the number of tokens that can be acted on. The `max_daily_amount` and `max_daily_transactions` input parameters to the `createAccount` method control this behavior. These parameters are optional.

You can achieve higher throughput if you do not set the daily transaction limits for an account.

createAccount (TypeScript)

```
@Validator(yup.string(), yup.string(), yup.string(), yup.object().nullable())

public async createAccount(org_id: string, user_id: string, token_type: 
string, daily_limits: DailyLimits) {
await this.Ctx.Auth.checkAuthorization("ACCOUNT.createAccount", "TOKEN", 
{ org_id });
return await this.Ctx.Account.createAccount(org_id, user_id, token_type, 
daily_limits);
}
```

Additional Parameters:

- `daily_limits: JSON` – An object specifying the maximum amount of tokens that can be used in transactions daily (`max_daily_amount`) and the maximum number of transactions that can be completed daily (`max_daily_transactions`) as shown in the following example.

```
{
     "max_daily_amount": 100000
     "max_daily_transactions": 10000
 }
```

CreateAccount (Go)

```
func (t *Controller) CreateAccount(org_id string, user_id string, token_type 
string, daily_limits ...account.AccountDailyLimits) (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, 
daily_limits...)
}
```

Additional Parameters:

- `daily_limits: JSON` – A JSON object that includes a `MaxDailyAmount` parameter (the maximum amount of tokens that can be used in transactions daily) and a `MaxDailyTransactions` parameter (the maximum number of transactions that can be completed daily), as shown in the following example.

```
{
     "MaxDailyAmount": 100000
     "MaxDailyTransactions": 10000
 }
```

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~a73085a385bc96c4a45aa2dff032e7dede82c0664dee5f396b7c5854
eeafd4bd",
   "BapAccountVersion": 0,
   "UserId":"user1",
   "OrgId":"Org1MSP",
   "AccountType":"fungible",
   "TokenId":"",
   "TokenName":"",
   "Balance":0,
   "BalanceOnHold":0
}
```

Approval Requirements for Minting and Burning

You can set up approvals for minting and burning tokens, so that users with the minter or burner role must submit a request to an approver, instead of minting or burning tokens directly. Approvers can accept or reject requests to mint or burn tokens. To enable approvals for minting and burning, you use the `mint_approval_required` and `burn_approval_required` parameters. You must then also specify values for `mint_approver_role_name` and `burn_approval_role_name`, as shown in the following example.

```
behavior: # Token behaviors
          - divisible:
                decimal: 2
          - mintable:
                max_mint_quantity: 1000
                mint_approval_required: true
          - transferable
          - burnable
                burn_approval_required: true
          - holdable
          - roles:
                minter_role_name: minter
                notary_role_name: notary
                mint_approver_role_name: minter_notary
                burn_approver_role_name: burner_notary
```

The following methods support requesting, accepting, and rejecting approvals to mint and burn tokens.

TypeScript Methods for Minting and Burning Approval

`requestMint`
This method can be called by a minter to send a request to the minter notary to create a specified amount of tokens.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.number().positive(), yup.date(), yup.object().nullable())
```

```
public async requestMint( token_id: string, operation_id: string,
notary_org_id: string, notary_user_id: string, quantity: number,
time_to_expiration: Date, info_details?: InfoDetails) {

const token_asset = await this.getTokenObject(token_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, null, notary_account_id,
quantity, time_to_expiration, token_asset, HoldOperationType.MINT,
info_details);

}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `operation_id: string` – The unique operation ID that represents the mint request.
- `notary_org_id: string` – The membership service provider (MSP) ID of the minter notary who will process the request.
- `notary_user_id: string` – The user name or email ID of the minter notary who will process the request.
- `quantity: number` – The amount of tokens to mint.
- `time_to_expiration` – The time after which the minting request expires and is no longer valid.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) has successfully submitted request to mint
100 tokens",
}
```

`approveMint`
This method can be called by a minter notary to approve a minting request.

```
@Validator(yup.string(), yup.string())
public async approveMint(token_id: string, operation_id: string) {
const token_asset = await this.getTokenObject(token_id);
```

```
return await this.Ctx.Token.executeHold(operation_id, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `operation_id: string` – The unique operation ID that represents the mint request.

Return Value Example:

```
{
msg:
"Successfully minted 100 tokens to Account Id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

`rejectMint`
This method can be called by a minter notary to reject a minting request.

```
@Validator(yup.string(), yup.string())
public async rejectMint(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 to mint.
- `operation_id: string` – The unique operation ID that represents the mint request.

Return Value Example:

```
{
 msg: "Successfully rejected mint request with Operation Id 'operation1' to
mint 100 tokens of token id token"
}
```

`requestBurn`
This method can be called by a burner to send a request to the burner notary to destroy a specified amount of tokens.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.number().positive(), yup.date(), yup.object().nullable())

public async requestBurn( token_id: string, operation_id: string,
notary_org_id: string, notary_user_id: string, quantity: number,
time_to_expiration: Date, info_details?: InfoDetails ) {

const token_asset = await this.getTokenObject(token_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, null, notary_account_id,
quantity, time_to_expiration, token_asset, HoldOperationType.BURN, null,
description);
}
```

Parameters:

- `token_id: string` – The ID of the token to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.
- `notary_org_id: string` – The membership service provider (MSP) ID of the burner notary who will process the request.
- `notary_user_id: string` – The user name or email ID of the burner notary who will process the request.
- `quantity: number` – The amount of tokens to burn.
- `time_to_expiration` – The time after which the burning request expires and is no longer valid.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) has successfully submitted request to mint
100 tokens",
}
```

`approveBurn`
This method can be called by a burner notary to approve a burning request.

```
@Validator(yup.string(), yup.string())
public async approveBurn(token_id: string, operation_id: string) {
const token_asset = await this.getTokenObject(token_id);
return await this.Ctx.Token.executeHold(operation_id, token_asset);
}
```

Parameters:

- `token_id: string` – The ID of the token to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.

Return Value Example:

```
{
msg:
"Successfully burned 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

rejectBurn
This method can be called by a burner notary to reject a burning request.

```
@Validator(yup.string(), yup.string())
public async rejectBurn(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 to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.

Return Value Example:

```
{
 msg: "Successfully rejected burn request with Operation Id 'operation1' to
burn 100 tokens of token id token",
}
```

Go Methods for Minting and Burning Approval

RequestMint
This method can be called by a minter to send a request to the minter notary to create a specified amount of tokens.

```
func (t *Controller) RequestMint(token_id string, operation_id string,
notary_org_id string, notary_user_id string, quantity float64,
timeToExpiration string, info_details ...token.InfoDetails) (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())
 }
return t.Ctx.Token.Hold(operation_id, "", notary_account_id, quantity,
timeToExpiration, tokenAssetValue.Interface(), constants.HoldMint,
info_details...)
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `operation_id: string` – The unique operation ID that represents the mint request.
- `notary_org_id: string` – The membership service provider (MSP) ID of the minter notary who will process the request.
- `notary_user_id: string` – The user name or email ID of the minter notary who will process the request.
- `quantity: number` – The amount of tokens to mint.
- `TimeToExpiration` – The time after which the minting request expires and is no longer valid.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "Category" : "category input",
       "Description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(org_id: Org1MSP, user_id: admin) has successfully submitted request to mint
100 tokens",
}
```

`ApproveMint`
This method can be called by a minter notary to approve a minting request.

```
func (t *Controller) ApproveMint(token_id string, operation_id string)
(interface{}, error) {
tokenAssetValue, err := t.getTokenObject(token_id)
if err != nil {
return nil, err
}
return t.Ctx.Token.ExecuteHold(operation_id, tokenAssetValue.Interface())
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.

- `operation_id: string` – The unique operation ID that represents the mint request.

Return Value Example:

```
{
msg:
"Successfully minted 100 tokens to Account Id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(org_id: Org1MSP, user_id: admin)"
}
```

RejectMint
This method can be called by a minter notary to reject a minting request.

```
func (t *Controller) RejectMint(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 to mint.
- `operation_id: string` – The unique operation ID that represents the mint request.

Return Value Example:

```
{
 msg: "Successfully rejected mint request with Operation Id 'operation1' to
mint 100 tokens of token id token"
}
```

RequestBurn
This method can be called by a burner to send a request to the burner notary to destroy a specified amount of tokens.

```
func (t *Controller) RequestBurn(token_id string, operation_id string,
notary_org_id string, notary_user_id string, quantity float64,
timeToExpiration string, info_details ...token.InfoDetails) (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())
 }
return t.Ctx.Token.Hold(operation_id, "", notary_account_id, quantity,
timeToExpiration, tokenAssetValue.Interface(), constants.HoldBurn,
info_details...)
}
```

Parameters:

- `token_id: string` – The ID of the token to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.
- `notary_org_id: string` – The membership service provider (MSP) ID of the burner notary who will process the request.
- `notary_user_id: string` – The user name or email ID of the burner notary who will process the request.
- `quantity: number` – The amount of tokens to burn.
- `time_to_expiration` – The time after which the burning request expires and is no longer valid.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(org_id: Org1MSP, user_id: admin) has successfully submitted request to mint
100 tokens",
}
```

`ApproveBurn`
This method can be called by a burner notary to approve a burning request.

```
func (t *Controller) ApproveBurn(token_id string, operation_id string)
(interface{}, error) {
tokenAssetValue, err := t.getTokenObject(token_id)
if err != nil {
return nil, err
}
return t.Ctx.Token.ExecuteHold(operation_id, tokenAssetValue.Interface())
}
```

Parameters:

- `token_id: string` – The ID of the token to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.

Return Value Example:

```
{
msg:
"Successfully burned 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(org_id: Org1MSP, user_id: admin)"
}
```

`RejectBurn`
This method can be called by a burner notary to reject a burning request.

```
func (t *Controller) RejectBurn(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 to burn.
- `operation_id: string` – The unique operation ID that represents the burn request.

Return Value Example:

```
{
 msg: "Successfully rejected burn request with Operation Id 'operation1' to
burn 100 tokens of token id token",
}
```

Fetching Transaction History from the Rich History Database

You can synchronize data to the rich history database and then fetch the data using chaincode API calls. The following method, shown in TypeScript and in Go, fetchs transaction history from the rich history database. Before you can use these methods, you must run Oracle Autonomous Database with Oracle REST Data Services (ORDS) and OAuth enabled, as described in Oracle Database View Definitions for Wholesale CBDC.

`getAccountTransactionHistoryWithFiltersFromRichHistDB (TypeScript)`

```
@GetMethod()
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string(), yup.object().nullable())
public async getAccountTransactionHistoryWithFiltersFromRichHistDB(token_id:
string, org_id: string, user_id: string, custom_endpoint: string,
bearer_token: string, filters?: Filters) {
```

```
const account_id = await this.Ctx.Account.generateAccountId(token_id, org_id,
user_id);
await
this.Ctx.Auth.checkAuthorization("ACCOUNT.getAccountTransactionHistoryWithFilt
ers", "TOKEN", { account_id });
return await
this.Ctx.Account.getAccountTrxHistoryWithFiltersFromRichHistDB(account_id,
org_id, user_id.toLowerCase(), custom_endpoint, bearer_token, filters);
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.
- `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.
- `custom_endpoint` – The RESTful service endpoint of the rich history database.
- `bearer_token` – The access authorization token for the RESTful service endpoint.
- `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.

`GetAccountTransactionHistoryWithFiltersFromRichHistDB (Go)`

```
func (t *Controller)
GetAccountTransactionHistoryWithFiltersFromRichHistDB(token_id string, org_id
string, user_id string, custom_endPoint string, bearer_token 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.GetAccountTransactionHistoryWithFilters
", "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.GetAccountTransactionHistoryWithFiltersFromRichHistDB(account_id
, org_id, user_id, custom_endPoint, bearer_token, filters...)
return transactionArray, err
}
```

Parameters:

- `token_id: string` – The ID of the token to mint.

- `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.
- `custom_endpoint` – The RESTful service endpoint of the rich history database.
- `bearer_token` – The access authorization token for the RESTful service endpoint.
- `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.

Category and Description Attributes in Transaction Objects

- Category and description attributes must be included in the `transferTokens`, `holdTokens`, `issueTokens`, `requestMint`, `requestBurn`, `burnTokens` and `rejectBurn` methods in the controller file. The corresponding SDK methods must also include category and description attributes.
- The category and description attribute input is in the form of a JSON object named `info_details`, as shown in the following example.

```
{
     "category" : "category input",
     "description" : "description input"
}
```

- The `info_details` field is optional. You can pass only a category or only a description as needed.
- The GET methods related to any transactions for `transferTokens`, `holdTokens`, `executeHold`, `releaseHold`, `requestMint`, `approveMint`, `rejectMint`, `requestBurn`, `approveBurn` and `rejectBurn` must include category and description attributes in the payload response if they are present.
- The category field is limited 20 characters and the description field is limited to 250 Characters.

TypeScript Methods with Modified Inputs

The following methods support optional category and description attributes when you use the enhanced version of Blockchain App Builder.

`transferTokens`
This method transfers tokens from the caller to a specified account.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.number().positive(),
yup.object().nullable())
public async transferTokens(token_id: string, to_org_id: string, to_user_id:
string, quantity: number, info_details?: InfoDetails) {
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,
info_details);
}
```

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.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
 msg: "Successfully transferred 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) to account id:
oaccount~7yuijg39b4e1e4136dd86a806020c97a930909325340481b8fdhjklliugbv699
(Org-Id: Org1MSP, User-Id: user)",
}
```

holdTokens
This method creates a hold on behalf of the owner of the tokens with the `to_account_id` account.

```
@Validator(yup.string(), yup.string(), yup.string(), yup.string(),
yup.string(), yup.string(), yup.number().positive(), yup.date(),
yup.object().nullable())
  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, info_details?: InfoDetails) {
    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,
HoldOperationType.TRANSFER, info_details);
  }
```

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`.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

```
{
     "category" : "category input",
     "description" : "description input"
}
```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) is successfully holding 100 tokens",
}
```

`issueTokens`
This method mints tokens, which are then owned by the caller of the method.

```
@Validator(yup.string(), yup.number().positive(), yup.object().nullable())
public async issueTokens(token_id: string, quantity: number, info_details?:
InfoDetails) {
const token_asset = await this.getTokenObject(token_id);
return await this.Ctx.Token.mint(quantity, token_asset, info_details);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to mint.

- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"Successfully minted 100 tokens to Account Id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

`burnTokens`
This method deactivates, or burns, tokens from the transaction caller's account.

```
@Validator(yup.string(), yup.number().positive(), yup.object().nullable())

public async burnTokens(token_id: string, quantity: number, info_details?:
InfoDetails) {
const token_asset = await this.getTokenObject(token_id);
return await this.Ctx.Token.burn(quantity, token_asset, info_details);
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to burn.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"Successfully burned 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

Go Methods with Modified Inputs

The following methods support optional category and description attributes when you use the enhanced version of Blockchain App Builder.

`TransferTokens`
This method transfers tokens from the caller to a specified account.

```
func (t *Controller) TransferTokens(token_id string, to_org_id string,
to_user_id string, quantity float64, info_details ...token.InfoDetails)
(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(), info_details...)
}
```

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.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
 msg: "Successfully transferred 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) to account id:
oaccount~7yuijg39b4e1e4136dd86a806020c97a930909325340481b8fdhjklliugbv699
(Org-Id: Org1MSP, User-Id: user)",
}
```

`HoldTokens`
This method creates a hold on behalf of the owner of the tokens with the `to_account_id` 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,
info_details ...token.InfoDetails) (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(),
constants.HoldTransfer, info_details...)
}
```

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`.

- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"AccountId
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin) is successfully holding 100 tokens",
}
```

`IssueTokens`
This method mints tokens, which are then owned by the caller of the method.

```
func (t *Controller) IssueTokens(token_id string, quantity float64,
info_details ...token.InfoDetails) (interface{}, error) {
tokenAssetValue, err := t.getTokenObject(token_id)
if err != nil {
return nil, err
 }
return t.Ctx.Token.Mint(quantity, tokenAssetValue.Interface(),
info_details...)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to mint.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"Successfully minted 100 tokens to Account Id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

`BurnTokens`
This method deactivates, or burns, tokens from the transaction caller's account.

```
func (t *Controller) BurnTokens(token_id string, quantity float64,
info_details ...token.InfoDetails) (interface{}, error) {
tokenAssetValue, err := t.getTokenObject(token_id)
if err != nil {
return nil, err
 }
return t.Ctx.Token.Burn(quantity, tokenAssetValue.Interface(),
info_details...)
}
```

Parameters:

- `token_id: string` – The ID of the token.
- `quantity` – The number of tokens to burn.
- `info_details: JSON` – An object specifying the category (`category`) and description (`description`) of the request, as shown in the following example.

  ```
  {
       "category" : "category input",
       "description" : "description input"
  }
  ```

Return Value Example:

```
{
msg:
"Successfully burned 100 tokens from account id:
oaccount~95be539b4e1e4136dd86a806020c97a930909325340481b8fd88d339874fa699
(Org-Id: Org1MSP, User-Id: admin)"
}
```

TypeScript Methods with Modified Outputs

The following methods return the relevant organization and user IDs when you use the enhanced version of Blockchain App Builder.

`getAccountTransactionHistory`
This method returns an array of account transaction history details for a specified user and token.

```
@GetMethod()
@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.

Return Value Example:

```
[
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
 ]
```

getAccountTransactionHistoryWithFilters
This method returns a filtered array of account transaction history details for a specified user and token.

```
@GetMethod()
@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.getAccountTransactionHistoryWithFilt
ers", "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.

Return Value Example:

```
[
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id": 
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
 ]
```

Go Methods with Modified Outputs

The following methods return the relevant organization and user IDs when you use the enhanced version of Blockchain App Builder.

`GetAccountTransactionHistory`
This method returns an array of account transaction history details for a specified user and token.

```
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.

Return Value Example:

```
[
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
 ]
```

GetAccountTransactionHistoryWithFilters

This method returns a filtered array of account transaction history details for a specified user and token.

```
func (t *Controller) GetAccountTransactionHistoryWithFilters(token_id string,
filters ...account.AccountHistoryFilters) (interface{}, error) {
org_id, err := t.Ctx.Model.GetTransientMapKeyAsString(constants.OrgIdCC)
if err != nil {
return nil, err
}
user_id, err := t.Ctx.Model.GetTransientMapKeyAsString(constants.UserIdCC)
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("Account.GetAccountTransactionHistoryWithFilters
", "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.GetReconciledTransactionHistory(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.

Return Value Example:

```
[
            {
                "transaction_id":
"otransaction~64c5a4830949eae1424600f3d4a438c6f603a7c3ea31a68e374b899803999e22
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:37:28.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REJECT_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~a4537ef34a955b023b7c205b9abf06a6c79e4fdd761fb24f41b8eb34126b66c0
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:32.000Z",
                "balance": 550,
                "onhold_balance": 10,
                "token_id": "USD",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "APPROVE_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~6237a759422bd9fb112742e8cd7e6450df5a74a32236d9b1005571afed8904a4
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:36:18.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            },
            {
                "transaction_id":
"otransaction~06b35071415d74aa1a7c18449149c937d886cae76a832c44cf8d98e84586e76e
",
                "transacted_amount": 10,
                "timestamp": "2024-12-11T13:35:46.000Z",
                "balance": 540,
                "onhold_balance": 10,
                "token_id": "USD",
                "category": "category value",
                "description": "description value",
                "transacted_account":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "transaction_type": "REQUEST_MINT",
                "transacted_org_id": "CB",
                "transacted_user_id'": "creator_user_cb"
            }
 ]
```

Bond Marketplace Model

The enhanced version of Blockchain App Builder includes a model attribute that generates additional methods for the bond marketplace scenario.

If you include the `model: bond` parameter in the specification file for tokens that use the extended ERC-1155 standard, Blockchain App Builder application-specific chaincode, including the following additional methods for use with the bond marketplace application.

TypeScript Methods for Bond Marketplace

The bond marketplace chaincode includes all methods available in the generic ERC-1155 NFT chaincode. The following additional methods that are specific to the bond marketplace scenario are available.

`postToken`
This method can be called only by a token creator. The method submits the bond token for listing in the marketplace. When a token is created, its status is initially set to `created`. This method updates the status to `posted`. Users can run the `getAllTokensWithFilter` method to retrieve all NFTs with a `posted` status.

```
public async postToken(tokenId: string)
```

Parameters:

- `tokenId: string` – The ID of the token to post.

Return Value Example:

```
{
         "isValid":true,
         "payload":{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"created"
         },
         "message":"Successfully updated asset with ID bond1"
      }
```

`purchaseToken`

This method can be called by any account holder to purchase a listed bond NFT. The purchase transfers the bond NFT from the creator's account to the caller's account, and transfers CBDC tokens from the caller's account to the creator's account. Because of this, the method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring that the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token creator, and the CBDC token transfer value must be equal to the face value of the bond token multiplied by the quantity being purchased.

```
public async purchaseToken(tokenId: string, quantity: number, orderId:
string, additionalFees: number)
```

Parameters:

- `tokenId: string` – The ID of the token to purchase.
- `quantity: number` – The amount of tokens to purchase.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the purchase price.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

payInterest
This method can be called only by the token creator or admin to pay the interest earned on the bond token. This method can be called only if the interest frequency of the token is monthly, quarterly, or annually. Interest cannot be paid if the interest frequency is at maturity. Interest is calculated by the chaincode based on the coupon rate of the token. The purchase operation transfers CBDC tokens from the caller's account to the bond owner’s account. Because of this, this method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token owner, and the CBDC token transfer value must be equal to the interest calculated by the bond chaincode.

```
public async payInterest(orgId: string, userId: string, tokenId: string,
orderId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
```

```
    }
}
```

`requestTokenRedemption`
This method can be called only by the token owner to raise a request for the redemption of bond tokens after maturity. This method is also involved the calculation of the redemption price by the chaincode. Redemption requests can be raised only on the entire quantity of the bond token that the user owns. Users can raise multiple redemption requests based on different settlement IDs but only one can be approved by the token creator.

```
public async requestTokenRedemption(settlementId: string, tokenId: string,
orderId: string, additionalFees: number)
```

Parameters:

- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the redemption price.

Return Value Example:

```
{
         "tokenName":"bond",
         "assetType":"otokenApproval",

"id":"otokenApproval~9e006057ac96ae997e3964531b1a08ad2316555701c7fe9ec7b88e38e
20892bf",
         "settlementId":"op4",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op4",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
         "tokenId":"bond1",
         "quantity":1,
         "status":"PENDING",
         "orderId":"op4",
         "redeemPrice":11

}
```

`approveTokenRedemption`
This method can be called only by the token creator to approve a request for the redemption of bond tokens. The approval operation transfers the bond NFT from the owner's account (the user who raised the request) to the creator's account, and transfers CBDC tokens from the bond creator's to the owner's account. Because of this, this method must be executed in the context of an atomic transaction. The method also verifies the transfer process, ensuring the

appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the account of the token owner who raised the redemption request, and the CBDC token transfer value must be equal to the calculated redemption price that was calculated by the chaincode while raising the redemption request.

```
public async approveRedemption(fromOrgId: string, fromUserId: string,
settlementId: string, tokenId: string)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{
            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

`rejectRedemption`
The token creator can call this method to reject the redemption request. Token owners can raise a redemption request again by using a different settlement ID.

```
public async rejectRedemption(fromOrgId: string, fromUserId: string, 
settlementId: string, tokenId: string)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
    "status":"success",
    "msg":"Successfully rejected the token approval request"
}
```

`getAllTokensWithFilters`
The admin can call this get method to fetch all the tokens filtered by status, either `CREATED` or `POSTED`.

```
public async getAllTokensWithFilters(status: string, pageSize: number, 
bookmark: string)
```

Parameters:

- `status: string` – The status of the token, which can either be `CREATED` or `POSTED`.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }

]
```

`getTokenApprovalRequestByUser`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
public async getTokenApprovalRequestByUser(status: string)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

`getTokenApprovalRequestForUserByStatus`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
public async getTokenApprovalRequestForUserByStatus(status: string)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

getAccountBondSummary
Any account holder can call this get method to get an account summary that includes the details of purchased or redeemed tokens and the purchase price and redemption price.

```
public async getAccountBondSummary(orgId: string, userId: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
{


"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",
"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"REJECTED"
      },
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "tokenId":"bond1",
         "status":"Purchased",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op2",
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"APPROVED"
      }
   ]
}
```

`getAccountBondSummaryWithPagination`

Any account holder can call this get method to get an account summary that includes details of purchased or redeemed tokens and the purchase price and redemption price. This method can return results with pagination based on pagesize and bookmark values, and also filtered by start time and end time.

```
public async getAccountBondSummaryWithPagination(orgId: string, userId:
string, pageSize: number, bookmark?: string)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
{

"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"REJECTED"
      },
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "tokenId":"bond1",
         "status":"Purchased",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op2",
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"APPROVED"
      }
   ]
}
```

Go Methods for Bond Marketplace

The bond marketplace chaincode includes all methods available in the generic ERC-1155 NFT chaincode. The following additional methods that are specific to the bond marketplace scenario are available.

`PostToken`
This method can be called only by a token creator. The method submits the bond token for listing in the marketplace. When a token is created, its status is initially set to `created`. This

method updates the status to `posted`. Users can run the `getAllTokensWithFilter` method to retrieve all NFTs with a `posted` status.

```
func (t *Controller) PostToken(tokenId string) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token to post.

Return Value Example:

```
{
         "isValid":true,
         "payload":{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "AssetType":"otoken",
            "Events":true,
            "TokenId":"bond1",
            "TokenName":"bond",
            "TokenDesc":"tokenDesc value",
            "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":0
            },
            "Quantity":100,

"CreatedBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "CreationDate":"2024-12-02T12:42:09.000Z",
            "Divisible":{
               "decimal":0
            },
            "IsBurned":false,
            "IsLocked":false,
            "TokenUri":"tokenUri value",
            "Status":"created"
         },
         "message":"Successfully updated asset with ID bond1"
      }
```

PurchaseToken
This method can be called by any account holder to purchase a listed bond NFT. The purchase transfers the bond NFT from the creator's account to the caller's account, and transfers CBDC tokens from the caller's account to the creator's account. Because of this, the method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring that the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token creator, and the CBDC token transfer value must be equal to the face value of the bond token multiplied by the quantity being purchased.

```
func (t *Controller) PurchaseToken(tokenId string, quantity float64, orderId
string, additionalFees float64) (interface{}, error)
```

Parameters:

- `tokenId: string` – The ID of the token to purchase.
- `quantity: number` – The amount of tokens to purchase.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the purchase price.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{
            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

PayInterest
This method can be called only by the token creator or admin to pay the interest earned on the bond token. This method can be called only if the interest frequency of the token is monthly, quarterly, or annually. Interest cannot be paid if the interest frequency is at maturity. Interest is calculated by the chaincode based on the coupon rate of the token. The purchase operation transfers CBDC tokens from the caller's account to the bond owner’s account. Because of this, this method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token owner, and the CBDC token transfer value must be equal to the interest calculated by the bond chaincode.

```
func (t *Controller) PayInterestEarned(orgId string, userId string, tokenId
string, orderId string, additionalFees float64) (interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
      ],
      "lrc":{

      },
      "globalStatus":"Success",
      "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
      "txStartTime":"2024-12-05T12:01:21.881988035Z"
   }
}
```

`RequestTokenRedemption`
This method can be called only by the token owner to raise a request for the redemption of bond tokens after maturity. This method is also involved the calculation of the redemption price by the chaincode. Redemption requests can be raised only on the entire quantity of the

bond token that the user owns. Users can raise multiple redemption requests based on different settlement IDs but only one can be approved by the token creator.

```
func (t *Controller) RequestTokenRedemption(settlementId string, tokenId
string, orderId string, additionalFees float64) (interface{}, error)
```

Parameters:

- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the redemption price.

Return Value Example:

```
{
         "TokenName":"bond",
         "AssetType":"otokenApproval",

"Id":"otokenApproval~9e006057ac96ae997e3964531b1a08ad2316555701c7fe9ec7b88e38e
20892bf",
         "SettlementId":"op4",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op4",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
         "TokenId":"bond1",
         "Quantity":1,
         "Status":"PENDING",
         "OrderId":"op4",
         "RedeemPrice":11
}
```

`ApproveTokenRedemption`
This method can be called only by the token creator to approve a request for the redemption of bond tokens. The approval operation transfers the bond NFT from the owner's account (the user who raised the request) to the creator's account, and transfers CBDC tokens from the bond creator's to the owner's account. Because of this, this method must be executed in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the account of the token owner who raised the

redemption request, and the CBDC token transfer value must be equal to the calculated redemption price that was calculated by the chaincode while raising the redemption request.

```
func (t *Controller) ApproveRedemption(fromOrgId string, fromUserId string,
settlementId string, tokenId string) (interface{}, error)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "returnCode":"Success",
   "error":"",
   "result":{
      "transactions":[
         {
            "channel":"test",
            "chaincode":"BondMarketplace",
            "txstatus":"Committed",
            "prepare":{

"txid":"e969f962df5efda2ea6287380e308cc974efd79dfff3567840ed3844bf936160"
            },
            "commit":{

"txid":"5544e928d3242291fb39189e8329679a9c81d61d6f72db60ca89135cd20fffef"
            },
            "rollback":{

            }
         },
         {
            "channel":"cbdctest",
            "chaincode":"cbdc",
            "txstatus":"Committed",
            "prepare":{

"txid":"1245885b1a0c7f12c41fa2f2905549b8a5f37ab3a5e094b9dca122cb0611a117"
            },
            "commit":{

"txid":"3c83e20c7d470cdc9c1b0e2e0ea8d9962d58ada8d1b8f0d2606c8aa1f0ae7741"
            },
            "rollback":{

            }
         }
```

```
        ],
        "lrc":{

        },
        "globalStatus":"Success",
        "globalTxid":"761bb7cc-1d66-4645-aeb2-50e4dbd23d83",
        "txStartTime":"2024-12-05T12:01:21.881988035Z"
    }
}
```

`RejectRedemption`
The token creator can call this method to reject the redemption request. Token owners can raise a redemption request again by using a different settlement ID.

```
func (t *Controller) RejectRedemption(fromOrgId string, fromUserId string,
settlementId string, tokenId string) (interface{}, error)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully rejected the token approval request"
}
```

`GetAllTokensWithFilter`
The admin can call this get method to fetch all the tokens filtered by status, either `CREATED` or `POSTED`.

```
func (t *Controller) GetAllTokensWithFilters(status string, pageSize int32,
bookmark string)
```

Parameters:

- `status: string` – The status of the token, which can either be `CREATED` or `POSTED`.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
[ {

               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "AssetType":"otoken",
            "Events":true,
            "TokenId":"bond1",
            "TokenName":"bond",
            "TokenDesc":"tokenDesc value",
            "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":0
            },
            "Quantity":100,

"CreatedBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "CreationDate":"2024-12-02T12:42:09.000Z",
            "Divisible":{
               "decimal":0
            },
            "IsBurned":false,
            "IsLocked":false,
            "TokenUri":"tokenUri value",
            "Status":"posted"
         }

]
```

`GetTokenApprovalRequestByUser`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
func (t *Controller) GetTokenApprovalRequestByUser(status string)
(interface{}, error)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "SettlementId":"op1",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":2,
      "Status":"APPROVED",
      "OrderId":"op1",
      "RedeemPrice":1,
      "PurchasedPrice":11,
      "InterestEarned":0
   },
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "SettlementId":"op3",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":1,
      "Status":"APPROVED",
      "OrderId":"op3",
      "RedeemPrice":11,
      "PurchasedPrice":11,
      "InterestEarned":0
   }
]
```

GetTokenApprovalRequestForUserByStatus
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
func (t *Controller) GetTokenApprovalRequestForUserByStatus(status string)
(interface{}, error)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "SettlementId":"op1",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":2,
      "Status":"APPROVED",
      "OrderId":"op1",
      "RedeemPrice":1,
      "PurchasedPrice":11,
      "InterestEarned":0
   },
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",
"Id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "SettlementId":"op3",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":1,
      "Status":"APPROVED",
      "OrderId":"op3",
      "RedeemPrice":11,
      "PurchasedPrice":11,
      "InterestEarned":0
   }
]
```

GetAccountBondSummary

Any account holder can call this get method to get an account summary that includes the details of purchased or redeemed tokens and the purchase price and redemption price.

```
func (t *Controller) GetAccountBondSummary(orgId string, userId string)
(interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.

Return Value Example:

```
{


"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "TokenId":"bond1",
         "Status":"Redeemed",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op1"
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"REJECTED"
      },
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "TokenId":"bond1",
         "Status":"Purchased",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op2",
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"APPROVED"
      }
   ]
}
```

`GetAccountBondSummaryWithPagination`

Any account holder can call this get method to get an account summary that includes details of purchased or redeemed tokens and the purchase price and redemption price. This method can return results with pagination based on pagesize and bookmark values, and also filtered by start time and end time.

```
func (t *Controller) GetAccountBondSummaryWithPagination(orgId string, userId
string, pageSize int32, bookmark string) (interface{}, error)
```

Parameters:

- `orgId: string` – The membership service provider (MSP) ID of the user.
- `userId: string` – The user name or email ID of the user.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
{
```

```
"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "TokenId":"bond1",
         "Status":"Redeemed",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op1"
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"REJECTED"
      },
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "TokenId":"bond1",
         "Status":"Purchased",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op2",
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"APPROVED"
      }
   ]
}
```

TypeScript SDK Methods for Bond Marketplace

`payInterest`
The token creator or admin can call this method to pay the interest earned on a bond token. This method can be called only if the interest frequency of the token is monthly, quarterly, or annually. Interest cannot be paid if the interest frequency is at maturity. Interest is calculated by the chaincode itself based on the coupon rate of the token. The purchase operation transfers CBDC tokens from the caller's account to the bond owner’s account. Because of this, this method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct

`orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token owner, and the CBDC token transfer value must be equal to the interest that is calculated by the bond chaincode.

```
this.Ctx.ERC1155AccountBondSummary.payInterest(userAccountId, tokenId,
orderId)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.

Return Value Example:

```
{
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "interestEarned": 11,
         "interestEarnedDate": "2024-12-02T00:00:00.000Z"
}
```

`requestTokenRedemption`
This method can be called only by the token owner to raise a request for the redemption of bond tokens after maturity. This method is also involved the calculation of the redemption price by the chaincode. Redemption requests can be raised only on the entire quantity of the bond token that the user owns. Users can raise multiple redemption requests based on different settlement IDs but only one can be approved by the token creator.

```
this.Ctx.ERC1155TokenApproval.requestTokenRedemption(callerUserAccountId,
settlementId, tokenId, orderId, quantity, additionalFees);
```

Parameters:

- `callerUserAccountId: string` – The account ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.

- `additionalFees: number` – The additional fees to add to the redemption price.

Return Value Example:

```
{
         "tokenName":"bond",
         "assetType":"otokenApproval",

"id":"otokenApproval~9e006057ac96ae997e3964531b1a08ad2316555701c7fe9ec7b88e38e
20892bf",
         "settlementId":"op4",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op4",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
         "tokenId":"bond1",
         "quantity":1,
         "status":"PENDING",
         "orderId":"op4",
         "redeemPrice":11
}
```

`approveTokenRedemption`
This method can be called only by the token creator to approve a request for the redemption of bond tokens. The approval operation transfers the bond NFT from the owner's account (the user who raised the request) to the creator's account, and transfers CBDC tokens from the bond creator's to the owner's account. Because of this, this method must be executed in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the account of the token owner who raised the redemption request, and the CBDC token transfer value must be equal to the calculated redemption price that was calculated by the chaincode while raising the redemption request.

```
this.Ctx.ERC1155TokenApproval.approveTokenApprovalRequest(fromUserAccountId,
settlementId, tokenId)
```

Parameters:

- `fromUserAccountId: string` – The account ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully approved the token approval request"
}
```

`rejectRedemption`
The token creator can call this method to reject the redemption request. Token owners can raise a redemption request again by using a different settlement ID.

```
this.Ctx.ERC1155TokenApproval.rejectTokenApprovalRequest(fromUserAccountId, 
settlementId, tokenId)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully rejected the token approval request"
}
```

`getAllTokensWithFilter`
The admin can call this get method to fetch all the tokens filtered by status, either `CREATED` or `POSTED`.

```
this.Ctx.ERC1155Token.getAllTokensWithFilters(status, pageSize, bookmark)
```

Parameters:

- `status: string` – The status of the token, which can either be `CREATED` or `POSTED`.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
[{
            "tokenMetadata":{
               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "assetType":"otoken",
            "events":true,
            "tokenId":"bond1",
            "tokenName":"bond",
            "tokenDesc":"tokenDesc value",
            "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":0
            },
            "quantity":100,

"createdBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "creationDate":"2024-12-02T12:42:09.000Z",
            "divisible":{
               "decimal":0
            },
            "isBurned":false,
            "isLocked":false,
            "tokenUri":"tokenUri value",
            "status":"posted"
         }

]
```

`getTokenApprovalRequestByUser`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
this.Ctx.ERC1155TokenApproval.getAllTokenApprovalRequestByUserByStatus(status)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

`getTokenApprovalRequestForUserByStatus`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
this.Ctx.ERC1155TokenApproval.getAllTokenApprovalRequestForUserByStatus(status
)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "settlementId":"op1",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":2,
      "status":"APPROVED",
      "orderId":"op1",
      "redeemPrice":1,
      "purchasedPrice":11,
      "interestEarned":0
   },
   {
      "tokenName":"bond",
      "assetType":"otokenApproval",

"id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "settlementId":"op3",

"userBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"fromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"toAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "tokenId":"bond1",
      "quantity":1,
      "status":"APPROVED",
      "orderId":"op3",
      "redeemPrice":11,
      "purchasedPrice":11,
      "interestEarned":0
   }
]
```

`getAccountBondSummary`
Any account holder can call this get method to get an account summary that includes the details of purchased or redeemed tokens and the purchase price and redemption price.

```
this.Ctx.ERC1155AccountBondSummary.getAllAccountBondSummary(userAccountId)
```

Parameters:

- `userAccountId: string` – The account ID of the user.

Return Value Example:

```
{

"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "tokenId":"bond1",
         "status":"Redeemed",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op1"
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"REJECTED"
      },
      {
         "purchasedQuantity":1,
         "assetType":"oUserBondDetails",
"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "tokenId":"bond1",
         "status":"Purchased",
         "purchasedAmount":11,
         "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "orderId":"op2",
         "redeemPrice":11,
         "quantityRedeem":1,
         "redeemStatus":"APPROVED"
      }
   ]
}
```

getAccountBondSummaryWithPagination

Any account holder can call this get method to get an account summary that includes details of purchased or redeemed tokens and the purchase price and redemption price. This method can return results with pagination based on pagesize and bookmark values, and also filtered by start time and end time.

```
this.Ctx.ERC1155AccountBondSummary.getAllAccountBondSummaryWithPagination(user
AccountId, pageSize, bookmark)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
    {


"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
       "orgId":"BondMPTest",
       "userId":"u10",
       "accountSummary":[
          {
             "purchasedQuantity":1,
             "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
             "tokenId":"bond1",
             "status":"Redeemed",
             "purchasedAmount":11,
             "purchasedDate":"2024-12-02T00:00:00.000Z",
"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
             "orderId":"op1"
             "redeemPrice":11,
             "quantityRedeem":1,
             "redeemStatus":"REJECTED"
          },
          {
             "purchasedQuantity":1,
             "assetType":"oUserBondDetails",

"id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
             "tokenId":"bond1",
             "status":"Purchased",
             "purchasedAmount":11,
             "purchasedDate":"2024-12-02T00:00:00.000Z",

"purchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
             "orderId":"op2",
             "redeemPrice":11,
             "quantityRedeem":1,
             "redeemStatus":"APPROVED"
          }
       ]
    }
```

On success, it returns a message.

```
    {

"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
       "orgId":"BondMPTest",
       "userId":"u10",
       "accountSummary":[
          {
             "PurchasedQuantity":1,
             "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
             "TokenId":"bond1",
             "Status":"Redeemed",
             "PurchasedAmount":11,
             "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
             "OrderId":"op1"
             "RedeemPrice":11,
             "QuantityRedeem":1,
             "RedeemStatus":"REJECTED"
          },
          {
             "PurchasedQuantity":1,
             "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
             "TokenId":"bond1",
             "Status":"Purchased",
             "PurchasedAmount":11,
             "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
             "OrderId":"op2",
             "RedeemPrice":11,
             "QuantityRedeem":1,
             "RedeemStatus":"APPROVED"
          }
       ]
    }
```

Go SDK Methods for Bond Marketplace

`PayInterest`

The token creator or admin can call this method to pay the interest earned on a bond token. This method can be called only if the interest frequency of the token is monthly, quarterly, or annually. Interest cannot be paid if the interest frequency is at maturity. Interest is calculated by the chaincode itself based on the coupon rate of the token. The purchase operation transfers CBDC tokens from the caller's account to the bond owner’s account. Because of this, this method must be run in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the token owner, and the CBDC token transfer value must be equal to the interest that is calculated by the bond chaincode.

```
t.Ctx.ERC1155AccountBondSummary.PayInterestEarned(userAccountId, tokenId,
orderId, additionalFees)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.

Return Value Example:

```
{
          "PurchasedQuantity":1,
          "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
          "TokenId":"bond1",
          "Status":"Redeemed",
          "PurchasedAmount":11,
          "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
          "OrderId":"op1"
          "InterestEarned": 11,
          "InterestEarnedDate": "2024-12-02T00:00:00.000Z"

}
```

`RequestTokenRedemption`
This method can be called only by the token owner to raise a request for the redemption of bond tokens after maturity. This method is also involved the calculation of the redemption price by the chaincode. Redemption requests can be raised only on the entire quantity of the bond token that the user owns. Users can raise multiple redemption requests based on different settlement IDs but only one can be approved by the token creator.

```
t.Ctx.ERC1155TokenApproval.RequestTokenRedemption(callerUserAccountId,
settlementId, tokenId, orderId, quantity, additionalFees)
```

Parameters:

- `callerUserAccountId: string` – The account ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.
- `orderId: string` – The order ID for the purchase operation.
- `additionalFees: number` – The additional fees to add to the redemption price.

Return Value Example:

```
{
          "TokenName":"bond",
          "AssetType":"otokenApproval",

"Id":"otokenApproval~9e006057ac96ae997e3964531b1a08ad2316555701c7fe9ec7b88e38e
20892bf",
          "SettlementId":"op4",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op4",
"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
         "TokenId":"bond1",
         "Quantity":1,
         "Status":"PENDING",
         "OrderId":"op4",
         "RedeemPrice":11
}
```

`ApproveTokenRedemption`
This method can be called only by the token creator to approve a request for the redemption of bond tokens. The approval operation transfers the bond NFT from the owner's account (the user who raised the request) to the creator's account, and transfers CBDC tokens from the bond creator's to the owner's account. Because of this, this method must be executed in the context of an atomic transaction. The method also verifies the transfer process, ensuring the appropriate CBDC chaincode is called with the correct `orgId` and `userId` for the transfer. The `orgId` and `userId` must correspond to the account of the token owner who raised the redemption request, and the CBDC token transfer value must be equal to the calculated redemption price that was calculated by the chaincode while raising the redemption request.

```
t.Ctx.ERC1155TokenApproval.ApproveTokenApprovalRequest(fromUserAccountId,
settlementId, tokenId)
```

Parameters:

- `fromUserAccountId: string` – The account ID of the user.
- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully approved the token approval request"
}
```

`RejectRedemption`
The token creator can call this method to reject the redemption request. Token owners can raise a redemption request again by using a different settlement ID.

```
t.Ctx.ERC1155TokenApproval.RejectTokenApprovalRequest(fromUserAccountId,
settlementId, tokenId)
```

Parameters:

- `fromOrgId: string` – The membership service provider (MSP) ID of the user.
- `fromUserId: string` – The user name or email ID of the user.

- `settlementId: string` – The settlement ID for the redemption operation.
- `tokenId: string` – The ID of the token.

Return Value Example:

```
{
   "status":"success",
   "msg":"Successfully rejected the token approval request"
}
```

`GetAllTokensWithFilter`
The admin can call this get method to fetch all the tokens filtered by status, either `CREATED` or `POSTED`.

```
t.Ctx.ERC1155Token.GetAllTokensWithFilters(status, pageSize, bookmark)
```

Parameters:

- `status: string` – The status of the token, which can either be `CREATED` or `POSTED`.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
[ {

               "ISIN":"ISIN value",
               "Segment":"Segment value",
               "Issuer":"Issuer value",
               "FaceValue":10,
               "IssueSize":999,
               "CouponRate":10,
               "InterestPaymentType":"simple",
               "InterestFrequency":"monthly",
               "IssueDate":"2023-03-28T15:16:36.000Z",
               "MaturityDate":"2023-03-28T15:16:36.000Z"
            },
            "AssetType":"otoken",
            "Events":true,
            "TokenId":"bond1",
            "TokenName":"bond",
            "TokenDesc":"tokenDesc value",
            "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":0
            },
            "Quantity":100,

"CreatedBy":"oaccount~276bcf1324b1ad1e493e22432db3b39f7a4b9bb17b8525c0391ea3ba
36138e00",
            "CreationDate":"2024-12-02T12:42:09.000Z",
            "Divisible":{
               "decimal":0
            },
            "IsBurned":false,
            "IsLocked":false,
            "TokenUri":"tokenUri value",
            "Status":"posted"
         }
]
```

`GetTokenApprovalRequestByUser`
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
t.Ctx.ERC1155TokenApproval.GetAllTokenApprovalRequestByUserByStatus(status)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "SettlementId":"op1",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":2,
      "Status":"APPROVED",
      "OrderId":"op1",
      "RedeemPrice":1,
      "PurchasedPrice":11,
      "InterestEarned":0
   },
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "SettlementId":"op3",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":1,
      "Status":"APPROVED",
      "OrderId":"op3",
      "RedeemPrice":11,
      "PurchasedPrice":11,
      "InterestEarned":0
   }
]
```

GetTokenApprovalRequestForUserByStatus
Any account holder can call this get method to get the details of all the token approval requests (redemption requests) they have made.

```
t.Ctx.ERC1155TokenApproval.GetAllTokenApprovalRequestForUserByStatus(status)
```

Parameters:

- `status: string` – The status of the request, which can be `PENDING`, `REJECTED`, or `APPROVED`.

Return Value Example:

```
[
    {
        "TokenName":"bond",
        "AssetType":"otokenApproval",

"Id":"otokenApproval~5b2a94283ae95e3d6e5b76ffd6f75b7bff231e4df270a82cdc1f6badd
17dea4b",
      "SettlementId":"op1",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op1",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":2,
      "Status":"APPROVED",
      "OrderId":"op1",
      "RedeemPrice":1,
      "PurchasedPrice":11,
      "InterestEarned":0
   },
   {
      "TokenName":"bond",
      "AssetType":"otokenApproval",

"Id":"otokenApproval~fdf28b2d271ac9c0fbd94a2dedbf365728c77795f3e931e5a4a2dcf48
039a989",
      "SettlementId":"op3",

"UserBondDetailsId":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff
17747ca25af3369e26289747~bond1~op3",

"FromAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",

"ToAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bfdab1411f670
2a507003a15",
      "TokenId":"bond1",
      "Quantity":1,
      "Status":"APPROVED",
      "OrderId":"op3",
      "RedeemPrice":11,
      "PurchasedPrice":11,
      "InterestEarned":0
   }
]
```

`GetAccountBondSummary`

Any account holder can call this get method to get an account summary that includes the details of purchased or redeemed tokens and the purchase price and redemption price.

```
t.Ctx.ERC1155AccountBondSummary.GetAllAccountBondSummary(userAccountId)
```

Parameters:

- `userAccountId: string` – The account ID of the user.

Return Value Example:

```
{


"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "TokenId":"bond1",
         "Status":"Redeemed",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op1"
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"REJECTED"
      },
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "TokenId":"bond1",
         "Status":"Purchased",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op2",
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"APPROVED"
      }
   ]
}
```

`GetAccountBondSummaryWithPagination`
Any account holder can call this get method to get an account summary that includes details of purchased or redeemed tokens and the purchase price and redemption price. This method

can return results with pagination based on pagesize and bookmark values, and also filtered by start time and end time.

```
t.Ctx.ERC1155AccountBondSummary.GetAllAccountBondSummaryWithPagination(userAcc
ountId, pageSize, bookmark)
```

Parameters:

- `userAccountId: string` – The account ID of the user.
- `pageSize: number` – The page size of the returned result.
- `bookmark: string` – The bookmark of the returned result.

Return Value Example:

```
{


"userAccountId":"ouaccount~df36ebaeb728c7768c0d5a3ecab435985c506ff57fd58221202
9b6f3c9cabf14",
   "orgId":"BondMPTest",
   "userId":"u10",
   "accountSummary":[
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op1",
         "TokenId":"bond1",
         "Status":"Redeemed",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op1"
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"REJECTED"
      },
      {
         "PurchasedQuantity":1,
         "AssetType":"oUserBondDetails",

"Id":"ouserbonddetails~ed3aaa9979bfe6302dcc83b1b903bd383fda60ff17747ca25af3369
e26289747~bond1~op2",
         "TokenId":"bond1",
         "Status":"Purchased",
         "PurchasedAmount":11,
         "PurchasedDate":"2024-12-02T00:00:00.000Z",

"PurchasedFromAccountId":"ouaccount~e76f696c0d6c626b24d35b3ac21de377a6da24c1bf
dab1411f6702a507003a15",
         "OrderId":"op2",
         "RedeemPrice":11,
         "QuantityRedeem":1,
         "RedeemStatus":"APPROVED"
      }
   ]
}
```

Wholesale CBDC Model

The enhanced version of Blockchain App Builder includes a model attribute that generates additional methods for the wholesale central bank digital currency (CBDC) scenario.

If you include the `model: wcbdc` parameter in the specification file for tokens that use the extended Token Taxonomy Framework standard, Blockchain App Builder application-specific chaincode, including the following additional methods and functionality for use with the wholesale CBDC application.

TypeScript Methods for Wholesale CBDC

The wholesale CBDC chaincode includes all methods available in the generic Token Taxonomy Framework NFT chaincode. The following additional methods that are specific to the wholesale CBDC scenario are available.

`setApplicationGroups`
This method sets the `application_groups` parameter in the account details for the specified application groups in the API. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

```
public async setApplicationGroups (org_id: string, user_id: string, token_id:
string, application_groups: 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_id: string` – The ID of the token.
- `application_groups: string[]` – A list of application groups the user ID belongs to, which define the user's associations in the CBDC application.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id": 
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "",
            "token_name": "",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-09T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 188
    }
}
```

`getAllActiveAccounts`
This method returns all of the active accounts that are associated with the specified token ID. Any user can call this method.

```
public async getAllActiveAccounts(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                "non_account_role_name": [
                    "token_admin"
                ],
                "role_name": null,
                "valueJson": {
                    "bapAccountVersion": 0,
                    "assetType": "oaccount",
                    "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                    "user_id": "admin_user_cb",
                    "org_id": "CB",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 0,
                    "onhold_balance": 0,
                    "application_groups": [
                        "CBDC_ADMINS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getAllSuspendedAccounts`
This method returns all of the suspended accounts that are associated with the specified token ID. Any user can call this method.

```
func (t *Controller) GetAllSuspendedAccounts(token_id string) (interface{}, 
error)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                "non_account_role_name": null,
                "role_name": null,
                "valueJson": {
                    "assetType": "oaccount",
                    "bapAccountVersion": 1,
                    "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                    "user_id": "user1_fi1",
                    "org_id": "FI1",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 5,
                    "onhold_balance": 0,
                    "application_groups": [
                        "FI_CBDC_USERS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getBurnQuantity`
This method returns the total quantity of burned tokens for a specified organization. This method can be called only by a `Token Admin`, `Token Auditor`, or a user with the burner role.

```
public async getBurnQuantity(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": {
          "burnt_quantity": 31
```

```
        },
        "encode": "JSON"
    }
}
```

getActionHistory
This method retrieves the history of approvals or rejections made by the caller for mint, burn, and transfer (issuance) operations, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
public async getActionHistory(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "from_account_id":
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "from_org_id": "CB",
              "from_user_id": "retirer_user_cb",
              "holding_id": "ohold~cbdc~USD~eaf6",
              "holding_status": "REJECT_BURN",
              "last_updated_time": "2024-11-26T21:43:22.000Z",
              "notary_account_id": null,
              "notary_org_id": null,
              "notary_user_id": null,
              "operation_id": null,
              "quantity": 3,
              "timetoexpiration": null,
              "to_account_id": "",
              "to_org_id": null,
              "to_user_id": null,
              "token_id": "USD",
              "token_name": null
          },
          {
              "from_account_id":
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "from_org_id": "CB",
              "from_user_id": "retirer_user_cb",
              "holding_id": "ohold~cbdc~USD~0031",
              "holding_status": "REJECT_BURN",
              "last_updated_time": "2024-11-26T21:43:15.000Z",
              "notary_account_id": null,
              "notary_org_id": null,
              "notary_user_id": null,
              "operation_id": null,
              "quantity": 2,
              "timetoexpiration": null,
              "to_account_id": "",
              "to_org_id": null,
              "to_user_id": null,
              "token_id": "USD",
              "token_name": null
          }
      ],
      "encode": "JSON"
  }
}
```

getPendingIssuance

This method retrieves all pending issuance (transfer) transactions where the caller is assigned as an approver, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary). This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `Notary`.

```
public async getPendingIssuance(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "from_org_id": "CB",
              "from_user_id": "creator_user_cb",
              "holding_id": "ohold~cbdc~USD~8e314",
              "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e314",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id":
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          },
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id":
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
              "from_org_id": "CB",
              "from_user_id": "issuer_user_cb",
              "holding_id": "ohold~cbdc~USD~8e315",
              "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e315",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id":
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          }
      ],
      "encode": "JSON"
  }
}
```

getPendingRequest
This method retrieves all pending requests of a specified type where the caller is assigned as an approver. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `Notary`.

```
public async getPendingRequest(token_id: string, request_type: string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `request_type: string` – The transaction type. For example, `mint` or `burn`.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "valueJson": {
                  "assetType": "ohold",
                  "holding_id": "ohold~cbdc~USD~op123",
                  "operation_id": "op123",
                  "token_id": "USD",
                  "token_name": "cbdc",
                  "operation_type": "mint",
                  "status": "pending",
                  "from_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_id": "",
                  "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
                  "quantity": 10,
                  "time_to_expiration": "0",
                  "category": "category value",
                  "description": "description value"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getTotalBalanceByCallerOrgId`
This method retrieves the total balance of the caller's organization. It can be called by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or any account owner.

```
public async getTotalBalanceByCallerOrgId()
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "totalBalance": 704
          }
      ],
      "encode": "JSON"
  }
}
```

`getTransactionWithBlockNumber`
This method returns the details of the transaction for the specified transaction ID.

```
public async getTransactionWithBlockNumber(token_id: string, transaction_id: 
string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `transaction_id: string` – The ID of the transaction.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "blockNo": 82,
              "key": 
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
              "metadata": null,
              "txnNo": 0,
              "value": null,
              "valueJson": {
                  "assetType": "otransaction",
                  "transaction_id": 
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
                  "token_id": "USD",
                  "from_account_id": "",
                  "from_account_balance": 0,
                  "from_account_onhold_balance": 0,
                  "to_account_id": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_balance": 100,
                  "to_account_onhold_balance": 0,
                  "transaction_type": "REQUEST_MINT",
                  "amount": 200,
                  "timestamp": "2024-11-20T06:48:42.000Z",
                  "number_of_sub_transactions": 0,
                  "holding_id": "",
                  "sub_transaction": "false",
                  "description": ""
              }
          }
      ],
      "encode": "JSON"
  }
}
```

The following API is modified for the wholesale CBDC model.

`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 formed by concatenating the asset type and token ID and then creating a SHA-256 hash over a concatenation of the organization ID and user ID. This method can be called only by a `Token Admin` of the chaincode.

```
public async createAccount(org_id: string, user_id: string, token_type:
string, application_groups: string[], daily_limits?: DailyLimits)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `application_groups: string[]` – A list of application groups the user Id belongs to, which define the user's associations in the CBDC application.
- `daily_limits: DailyLimits` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
```

```
                "token_id": "",
                "token_name": "",
                "balance": 0,
                "onhold_balance": 0,
                "application_groups": [
                    "CBDC_ADMINS"
                ],
                "max_daily_amount": 10000,
                "daily_amount": 0,
                "max_daily_transactions": 100,
                "daily_transactions": 0,
                "current_date": "2024-12-09T00:00:00.000Z"
            },
            "encode": "JSON",
            "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
            "blockNumber": 188
        }
}
```

Go Methods for Wholesale CBDC

The wholesale CBDC chaincode includes all methods available in the generic Token Taxonomy Framework NFT chaincode. The following additional methods that are specific to the wholesale CBDC scenario are available.

`SetApplicationGroups`
This method sets the `application_groups` parameter in the account details for the specified application groups in the API. This method can be called only by a `Token Admin` or `Org Admin` of the specified organization.

```
func (t *Controller) SetApplicationGroups(token_id string, org_id string,
user_id string, application_groups []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.
- `token_id: string` – The ID of the token.
- `application_groups: string[]` – A list of application groups the user ID belongs to, which define the user's associations in the CBDC application.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"261c3da4bb6f1724bc8f674c7b001a9b986bc9900d0508363039424926b143ed",
        "payload": {
            "AssetType": "oaccount",
            "AccountId":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "UserId": "admin_user_cb",
            "OrgId": "CB",
            "TokenType": "fungible",
            "TokenId": "",
            "TokenName": "",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "ApplicationGroups": [
                "CBDC_ADMINS"
            ],
            "MaxDailyAmount": 10000,
            "DailyAmount": 0,
            "MaxDailyTransactions": 100,
            "DailyTransactions": 0,
            "CurrentDate": "2024-12-09"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 246
    }
}
```

`GetAllActiveAccounts`
This method returns all of the active accounts that are associated with the specified token ID. Any user can call this method.

```
func (t *Controller) GetAllActiveAccounts(token_id string) (interface{},
error)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "NonAccountRoleName": "[\"token_admin\"]",
                "RoleName": null,
                "key":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                "valueJson": {
                    "AccountId":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                    "ApplicationGroups": [
                        "CBDC_ISSUERS"
                    ],
                    "AssetType": "oaccount",
                    "Balance": 0,
                    "BalanceOnHold": 0,
                    "BapAccountVersion": 0,
                    "CurrentDate": "2024-11-21",
                    "DailyAmount": 0,
                    "DailyTransactions": 0,
                    "MaxDailyAmount": 10000,
                    "MaxDailyTransactions": 100,
                    "OrgId": "CB",
                    "TokenId": "USD",
                    "TokenName": "cbdc",
                    "TokenType": "fungible",
                    "UserId": "admin_user_cb"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": "[\"token_auditor\"]",
              "RoleName": null,
              "key":
"oaccount~bd6262ffdf582675dd9b2506c1d5488864feef0b9e297a9a3322b7c683ad6214",
              "valueJson": {
                  "AccountId":
"oaccount~bd6262ffdf582675dd9b2506c1d5488864feef0b9e297a9a3322b7c683ad6214",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "auditor_user_cb",
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": "[\"token_admin\"]",
              "RoleName": null,
              "key":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
              "valueJson": {
                  "AccountId":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "admin_user_cb"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`GetAllSuspendedAccounts`
This method returns all of the suspended accounts that are associated with the specified token ID. Any user can call this method.

```
public async getAllSuspendedAccounts(token_id: string)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": null,
              "RoleName": "minter",
              "key":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "valueJson": {
                  "AccountId":
```

```
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "ApplicationGroups": [
                      "CBDC_ADMINS"
                  ],
                  "AssetType": "oaccount",
                  "Balance": 140,
                  "BalanceOnHold": 0,
                  "BapAccountVersion": 8,
                  "CurrentDate": "2024-11-21",
                  "DailyAmount": 70,
                  "DailyTransactions": 3,
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "TokenName": "cbdc",
                  "TokenType": "fungible",
                  "UserId": "creator_user_cb"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": null,
              "RoleName": "minter",
              "key":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "valueJson": {
                  "AccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "creator_user_cb",
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "NonAccountRoleName": null,
                "RoleName": "minter",
                "key":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "valueJson": {
                    "AccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                    "OrgId": "CB",
                    "TokenId": "USD",
                    "UserId": "creator_user_cb"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

`GetBurnQuantity`
This method returns the total quantity of burned tokens for a specified organization. This method can be called only by a `Token Admin`, `Token Auditor`, or a user with the burner role.

```
func (t *Controller) GetBurnQuantity(token_id string) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": {
          "BurnQuantity": 10
      },
      "encode": "JSON"
  }
}
```

GetActionHistory
This method retrieves the history of approvals or rejections made by the caller for mint, burn, and transfer (issuance) operations, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
func (t *Controller) GetActionHistory(token_id string) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "FromAccountId":
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "FromOrgId": "CB",
              "FromUserId": "retirer_user_cb",
              "HoldingId": "ohold~cbdc~USD~6e1223",
              "HoldingStatus": "REJECT_BURN",
              "LastUpdatedTime": "2024-11-21T22:08:26Z",
              "NotaryAccountId": null,
              "NotaryOrgId": null,
              "NotaryUserId": null,
              "OperationId": null,
              "Quantity": 5,
              "TimeToExpiration": null,
              "ToAccountId": "",
              "ToOrgId": null,
              "ToUserId": null,
              "TokenId": "USD",
              "TokenName": null
          },
          {
              "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "FromOrgId": "CB",
              "FromUserId": "creator_user_cb",
              "HoldingId": "ohold~cbdc~USD~hold2",
              "HoldingStatus": "RELEASEHOLD",
              "LastUpdatedTime": "2024-11-21T21:54:33Z",
              "NotaryAccountId": null,
              "NotaryOrgId": null,
              "NotaryUserId": null,
              "OperationId": null,
              "Quantity": 10,
              "TimeToExpiration": null,
              "ToAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "ToOrgId": "CB",
              "ToUserId": "creator_user_cb",
              "TokenId": "USD",
              "TokenName": null
          }
      ],
      "encode": "JSON"
  }
}
```

GetPendingIssuance

This method retrieves all pending issuance (transfer) transactions where the caller is assigned as an approver, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary). This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `Notary`.

```
func (t *Controller) GetPendingIssuance(token_id string) (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "AssetType": "ONHOLD",
              "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "FromOrgId": "CB",
              "FromUserId": "creator_user_cb",
              "HoldingId": "ohold~cbdc~USD~h123",
              "NotaryAccountId":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "NotaryOrgId": "CB",
              "NotaryUserId": "manager_user_cb",
              "OperationId": "h123",
              "Quantity": 10,
              "TimeToExpiration": "0",
              "ToAccountId":
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
              "ToOrgId": "CB",
              "ToUserId": "issuer_user_cb",
              "TokenId": "USD",
              "TokenName": "cbdc"
          }
      ],
      "encode": "JSON"
  }
}
```

GetPendingRequest

This method retrieves all pending requests of a specified type where the caller is assigned as an approver. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode, an `Org Admin` or `Org Auditor` of the specified organization, or the `Notary`.

```
func (t *Controller) GetPendingRequest(token_id string, request_type string)
(interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.
- `request_type: string` – The transaction type. For example, `mint` or `burn`.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "valueJson": {
                  "AssetType": "ohold",
                  "Category": "Category value",
                  "Description": "Description value",
                  "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "HoldingId": "ohold~cbdc~USD~8e1232",
                  "NotaryAccountId":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
                  "OperationId": "8e1232",
                  "OperationType": "mint",
                  "Quantity": 100,
                  "Status": "pending",
                  "TimeToExpiration": "0",
                  "ToAccountId": "",
                  "TokenId": "USD",
                  "TokenName": "cbdc"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`GetTotalBalanceByCallerOrgId`
This method retrieves the total balance of the caller's organization. It can be called by a `Token Admin`, `Token Auditor`, `Org Admin`, `Org Auditor`, or any account owner.

```
func (t *Controller) GetTotalBalanceByCallerOrgId() (interface{}, error)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "TotalBalance": 180
          }
      ],
      "encode": "JSON"
  }
}
```

`GetTransactionWithBlockNumber`
This method returns the details of the transaction for the specified transaction ID.

```
func (t *Controller) GetTransactionWithBlockNumber(token_id string,
transaction_id string)
```

Parameters:

- `token_id: string` – The ID of the token.
- `transaction_id: string` – The ID of the transaction.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "blockNo": 152,
              "key":
"otransaction~eee2de20b4b042884da83e3b7b85d8532ad56f26a546ee25d227acce33375c1c
",
              "metadata": null,
              "txnNo": 0,
              "value": null,
              "valueJson": {
                  "Amount": 10,
                  "AssetType": "otransaction",
                  "Category": "Category value",
                  "Description": "",
                  "FromAccountBalance": 130,
                  "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "FromAccountOnHoldBalance": 10,
                  "HoldingId": "ohold~cbdc~USD~hold1",
                  "NumberOfSubTransactions": 0,
                  "SubTransaction": "false",
                  "SubTransactionType": "",
                  "Timestamp": "2024-11-21T20:43:59Z",
                  "ToAccountBalance": 10,
                  "ToAccountId":
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
                  "ToAccountOnHoldBalance": 0,
                  "TokenId": "USD",
                  "TransactionId":
"otransaction~eee2de20b4b042884da83e3b7b85d8532ad56f26a546ee25d227acce33375c1c
",
                  "TransactionType": "EXECUTEHOLD"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

The following API is modified for the wholesale CBDC model.

`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 formed by concatenating the asset type and token ID and then creating a SHA-256 hash over a concatenation of the organization ID and user ID. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) CreateAccount(org_id string, user_id string, token_type
string, application_groups []string,
daily_limits ...account.AccountDailyLimits) (interface{}, error)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.

- `application_groups: string[]` – A list of application groups the user Id belongs to, which define the user's associations in the CBDC application.
- `daily_limits: DailyLimits` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"261c3da4bb6f1724bc8f674c7b001a9b986bc9900d0508363039424926b143ed",
        "payload": {
            "AssetType": "oaccount",
            "AccountId": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "UserId": "admin_user_cb",
            "OrgId": "CB",
            "TokenType": "fungible",
            "TokenId": "",
            "TokenName": "",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "ApplicationGroups": [
                "CBDC_ADMINS"
            ],
            "MaxDailyAmount": 10000,
            "DailyAmount": 0,
            "MaxDailyTransactions": 100,
            "DailyTransactions": 0,
            "CurrentDate": "2024-12-09"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 246
    }
}
```

TypeScript SDK Methods for Wholesale CBDC

`setApplicationGroups`
This method sets the `application_groups` parameter in the account details for the specified application groups in the API.

```
this.Ctx.Account.setApplicationGroups (account_id, application_groups)
```

Parameters:

- `account_id: string` – The ID of the account.
- `application_groups: string[]` – A list of application groups the user ID belongs to, which define the user's associations in the CBDC application.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "",
            "token_name": "",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-09T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 188
    }
}
```

`getAllActiveAccounts`
This method returns all of the active accounts that are associated with the specified token ID.

```
this.Ctx.CBDCToken.getAllActiveAccounts(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                "non_account_role_name": [
                    "token_admin"
                ],
                "role_name": null,
                "valueJson": {
                    "bapAccountVersion": 0,
                    "assetType": "oaccount",
                    "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                    "user_id": "admin_user_cb",
                    "org_id": "CB",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 0,
                    "onhold_balance": 0,
                    "application_groups": [
                        "CBDC_ADMINS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
              "non_account_role_name": [
                  "token_admin"
              ],
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
                  "org_id": "CB",
                  "user_id": "admin_user_cb",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getAllSuspendedAccounts`
This method returns all of the suspended accounts that are associated with the specified token ID.

```
this.Ctx.CBDCToken.getAllSuspendedAccounts(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "key":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                "non_account_role_name": null,
                "role_name": null,
                "valueJson": {
                    "assetType": "oaccount",
                    "bapAccountVersion": 1,
                    "account_id":
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                    "user_id": "user1_fi1",
                    "org_id": "FI1",
                    "token_type": "fungible",
                    "token_id": "USD",
                    "token_name": "cbdc",
                    "balance": 5,
                    "onhold_balance": 0,
                    "application_groups": [
                        "FI_CBDC_USERS"
                    ],
                    "max_daily_amount": 10000,
                    "daily_amount": 0,
                    "max_daily_transactions": 1000,
                    "daily_transactions": 0,
                    "current_date": "2024-11-20T00:00:00.000Z"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD",
                  "max_daily_amount": 10000,
                  "max_daily_transactions": 1000
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "key": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
              "non_account_role_name": null,
              "role_name": null,
              "valueJson": {
                  "account_id": 
"oaccount~802bf8da5579c6103b2dddaa6c4385df8e722d639a18029e0e93d7a5d6f826d6",
                  "org_id": "FI1",
                  "user_id": "user1_fi1",
                  "token_id": "USD"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`getBurnQuantity`
This method returns the total quantity of burned tokens for a specified organization.

```
this.Ctx.CBDCToken.getBurnQuantity(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "burnt_quantity": 31
        },
        "encode": "JSON"
    }
}
```

`getActionHistory`
This method retrieves the history of approvals or rejections made by the caller for mint, burn, and transfer (issuance) operations, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
this.Ctx.CBDCToken.getActionHistory(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "from_account_id":
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
                "from_org_id": "CB",
                "from_user_id": "retirer_user_cb",
                "holding_id": "ohold~cbdc~USD~eaf6",
                "holding_status": "REJECT_BURN",
                "last_updated_time": "2024-11-26T21:43:22.000Z",
                "notary_account_id": null,
                "notary_org_id": null,
                "notary_user_id": null,
                "operation_id": null,
                "quantity": 3,
                "timetoexpiration": null,
                "to_account_id": "",
                "to_org_id": null,
                "to_user_id": null,
                "token_id": "USD",
                "token_name": null
            },
            {
                "from_account_id": 
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
                "from_org_id": "CB",
                "from_user_id": "retirer_user_cb",
                "holding_id": "ohold~cbdc~USD~0031",
                "holding_status": "REJECT_BURN",
                "last_updated_time": "2024-11-26T21:43:15.000Z",
                "notary_account_id": null,
                "notary_org_id": null,
                "notary_user_id": null,
                "operation_id": null,
                "quantity": 2,
                "timetoexpiration": null,
                "to_account_id": "",
                "to_org_id": null,
                "to_user_id": null,
                "token_id": "USD",
                "token_name": null
            }
        ],
        "encode": "JSON"
    }
}
```

getPendingIssuance

This method retrieves all pending issuance (transfer) transactions where the caller is assigned as an approver, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
this.Ctx.CBDCToken.getPendingIssuance(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "from_org_id": "CB",
              "from_user_id": "creator_user_cb",
              "holding_id": "ohold~cbdc~USD~8e314",
              "notary_account_id": 
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e314",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id": 
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          },
          {
              "asset_type": "ONHOLD",
              "category": "category value",
              "from_account_id": 
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
              "from_org_id": "CB",
              "from_user_id": "issuer_user_cb",
              "holding_id": "ohold~cbdc~USD~8e315",
              "notary_account_id": 
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "notary_org_id": "CB",
              "notary_user_id": "manager_user_cb",
              "operation_id": "8e315",
              "quantity": 10,
              "timetoexpiration": "0",
              "to_account_id": 
"oaccount~44b844deccc6c314e14b8b9b95b51db5c8de499dbdbd3def2a44ba54c899c142",
              "to_org_id": "FI1",
              "to_user_id": "officer_user1_fi1",
              "token_id": "USD",
              "token_name": "cbdc"
          }
      ],
      "encode": "JSON"
  }
}
```

getPendingRequest
This method retrieves all pending requests of a specified type where the caller is assigned as an approver.

```
this.Ctx.CBDCToken.getPendingRequest(token_id, request_type)
```

Parameters:

- `token_id: string` – The ID of the token.
- `request_type: string` – The transaction type. For example, `mint` or `burn`.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "valueJson": {
                  "assetType": "ohold",
                  "holding_id": "ohold~cbdc~USD~op123",
                  "operation_id": "op123",
                  "token_id": "USD",
                  "token_name": "cbdc",
                  "operation_type": "mint",
                  "status": "pending",
                  "from_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_id": "",
                  "notary_account_id":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
                  "quantity": 10,
                  "time_to_expiration": "0",
                  "category": "category value",
                  "description": "description value"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

getTotalBalanceByCallerOrgId
This method retrieves the total balance of the caller's organization.

```
this.Ctx.CBDCToken.getTotalBalanceByCallerOrgId()
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "totalBalance": 704
          }
      ],
      "encode": "JSON"
  }
}
```

`getTransactionWithBlockNumber`
This method returns the details of the transaction for the specified transaction ID.

```
this.Ctx.CBDCToken.getTransactionWithBlockNumber(token_id, transaction_id)
```

Parameters:

- `token_id: string` – The ID of the token.
- `transaction_id: string` – The ID of the transaction.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "blockNo": 82,
              "key":
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
              "metadata": null,
              "txnNo": 0,
              "value": null,
              "valueJson": {
                  "assetType": "otransaction",
                  "transaction_id":
"otransaction~24f391919a8837d654beaa7346148ea8b2b9704624aef482ce68078c485f5b1b
",
                  "token_id": "USD",
                  "from_account_id": "",
                  "from_account_balance": 0,
                  "from_account_onhold_balance": 0,
                  "to_account_id":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "to_account_balance": 100,
                  "to_account_onhold_balance": 0,
                  "transaction_type": "REQUEST_MINT",
                  "amount": 200,
                  "timestamp": "2024-11-20T06:48:42.000Z",
                  "number_of_sub_transactions": 0,
                  "holding_id": "",
                  "sub_transaction": "false",
                  "description": ""
              }
          }
      ],
      "encode": "JSON"
  }
}
```

The following SDK method is modified for the wholesale CBDC model.

`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 formed by concatenating the asset type and token ID and then creating a SHA-256 hash over a concatenation of the organization ID and user ID.

```
this.Ctx.Account.createAccount(org_id, user_id, token_type,
application_groups, daily_limits)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `application_groups: string[]` – A list of application groups the user Id belongs to, which define the user's associations in the CBDC application.
- `daily_limits: DailyLimits` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"453821c7ffd477987ef8ccbd836b893969531ab768098cd4a99e3b89cd38a391",
        "payload": {
            "bapAccountVersion": 0,
            "assetType": "oaccount",
            "account_id":
"oaccount~28ac774001f374064029d51af4fb67e26ea1ea9ef62828b7a72dbf3beb8efd8d",
            "user_id": "admin_user_cb",
            "org_id": "CB",
            "token_type": "fungible",
            "token_id": "",
            "token_name": "",
            "balance": 0,
            "onhold_balance": 0,
            "application_groups": [
                "CBDC_ADMINS"
            ],
            "max_daily_amount": 10000,
            "daily_amount": 0,
            "max_daily_transactions": 100,
            "daily_transactions": 0,
            "current_date": "2024-12-09T00:00:00.000Z"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 188
    }
}
```

Go SDK Methods for Wholesale CBDC

`SetApplicationGroups`

This method sets the `application_groups` parameter in the account details for the specified application groups in the API.

```
t.Ctx.CbdcToken.SetApplicationGroups(account_id, application_groups)
```

Parameters:

- `account_id: string` – The ID of the account.
- `application_groups: string[]` – A list of application groups the user ID belongs to, which define the user's associations in the CBDC application.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"261c3da4bb6f1724bc8f674c7b001a9b986bc9900d0508363039424926b143ed",
        "payload": {
            "AssetType": "oaccount",
            "AccountId":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "UserId": "admin_user_cb",
            "OrgId": "CB",
            "TokenType": "fungible",
            "TokenId": "",
            "TokenName": "",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "ApplicationGroups": [
                "CBDC_ADMINS"
            ],
            "MaxDailyAmount": 10000,
            "DailyAmount": 0,
            "MaxDailyTransactions": 100,
            "DailyTransactions": 0,
            "CurrentDate": "2024-12-09"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 246
    }
}
```

`GetAllActiveAccounts`
This method returns all of the active accounts that are associated with the specified token ID.

```
t.Ctx.CbdcToken.GetAllActiveAccounts(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "NonAccountRoleName": "[\"token_admin\"]",
                "RoleName": null,
                "key": 
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                "valueJson": {
                    "AccountId": 
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                    "ApplicationGroups": [
                        "CBDC_ISSUERS"
                    ],
                    "AssetType": "oaccount",
                    "Balance": 0,
                    "BalanceOnHold": 0,
                    "BapAccountVersion": 0,
                    "CurrentDate": "2024-11-21",
                    "DailyAmount": 0,
                    "DailyTransactions": 0,
                    "MaxDailyAmount": 10000,
                    "MaxDailyTransactions": 100,
                    "OrgId": "CB",
                    "TokenId": "USD",
                    "TokenName": "cbdc",
                    "TokenType": "fungible",
                    "UserId": "admin_user_cb"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": "[\"token_auditor\"]",
              "RoleName": null,
              "key": 
"oaccount~bd6262ffdf582675dd9b2506c1d5488864feef0b9e297a9a3322b7c683ad6214",
              "valueJson": {
                  "AccountId": 
"oaccount~bd6262ffdf582675dd9b2506c1d5488864feef0b9e297a9a3322b7c683ad6214",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "auditor_user_cb",
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": "[\"token_admin\"]",
              "RoleName": null,
              "key":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
              "valueJson": {
                  "AccountId":
"oaccount~8db15b42910eeec401e1bf22c69dfdd11c820ecc26539ea03a3426fa25cb8c28",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "admin_user_cb"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`GetAllSuspendedAccounts`
This method returns all of the suspended accounts that are associated with the specified token ID.

```
t.Ctx.CbdcToken.GetAllSuspendedAccounts(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Returns:

- On success, a message that includes user details. The output varies based on the user's role, as shown in the following examples.

Return Value Example (Token Admin, Token Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": null,
              "RoleName": "minter",
              "key": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "valueJson": {
                  "AccountId": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "ApplicationGroups": [
                      "CBDC_ADMINS"
                  ],
                  "AssetType": "oaccount",
                  "Balance": 140,
                  "BalanceOnHold": 0,
                  "BapAccountVersion": 8,
                  "CurrentDate": "2024-11-21",
                  "DailyAmount": 70,
                  "DailyTransactions": 3,
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "TokenName": "cbdc",
                  "TokenType": "fungible",
                  "UserId": "creator_user_cb"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (Organization Admin, Organization Auditor):

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "NonAccountRoleName": null,
              "RoleName": "minter",
              "key": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "valueJson": {
                  "AccountId": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "OrgId": "CB",
                  "TokenId": "USD",
                  "UserId": "creator_user_cb",
                  "MaxDailyAmount": 10000,
                  "MaxDailyTransactions": 100,
              }
          }
      ],
      "encode": "JSON"
  }
}
```

Return Value Example (all other users):

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": [
            {
                "NonAccountRoleName": null,
                "RoleName": "minter",
                "key":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                "valueJson": {
                    "AccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                    "OrgId": "CB",
                    "TokenId": "USD",
                    "UserId": "creator_user_cb"
                }
            }
        ],
        "encode": "JSON"
    }
}
```

`GetBurnQuantity`
This method returns the total quantity of burned tokens for a specified organization.

```
t.Ctx.CbdcToken.GetBurnQuantity(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": {
          "BurnQuantity": 10
      },
      "encode": "JSON"
  }
}
```

`GetActionHistory`
This method retrieves the history of approvals or rejections made by the caller for mint, burn, and transfer (issuance) operations, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
t.Ctx.CbdcToken.GetActionHistory(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "FromAccountId":
"oaccount~0d7b3f73aea28065017ce8b79c0bb19256dc0cb475a0b2a85192bd110f69535c",
              "FromOrgId": "CB",
              "FromUserId": "retirer_user_cb",
              "HoldingId": "ohold~cbdc~USD~6e1223",
              "HoldingStatus": "REJECT_BURN",
              "LastUpdatedTime": "2024-11-21T22:08:26Z",
              "NotaryAccountId": null,
              "NotaryOrgId": null,
              "NotaryUserId": null,
              "OperationId": null,
              "Quantity": 5,
              "TimeToExpiration": null,
              "ToAccountId": "",
              "ToOrgId": null,
              "ToUserId": null,
              "TokenId": "USD",
              "TokenName": null
          },
          {
              "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "FromOrgId": "CB",
              "FromUserId": "creator_user_cb",
              "HoldingId": "ohold~cbdc~USD~hold2",
              "HoldingStatus": "RELEASEHOLD",
              "LastUpdatedTime": "2024-11-21T21:54:33Z",
              "NotaryAccountId": null,
              "NotaryOrgId": null,
              "NotaryUserId": null,
              "OperationId": null,
              "Quantity": 10,
              "TimeToExpiration": null,
              "ToAccountId": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "ToOrgId": "CB",
              "ToUserId": "creator_user_cb",
              "TokenId": "USD",
              "TokenName": null
          }
      ],
      "encode": "JSON"
  }
}
```

`GetPendingIssuance`

This method retrieves all pending issuance (transfer) transactions where the caller is assigned as an approver, including details of the organization, and user IDs of accounts involved (sender, recipient, and notary).

```
t.Ctx.CbdcToken.GetPendingIssuance(token_id)
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "AssetType": "ONHOLD",
              "FromAccountId": 
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
              "FromOrgId": "CB",
              "FromUserId": "creator_user_cb",
              "HoldingId": "ohold~cbdc~USD~h123",
              "NotaryAccountId": 
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
              "NotaryOrgId": "CB",
              "NotaryUserId": "manager_user_cb",
              "OperationId": "h123",
              "Quantity": 10,
              "TimeToExpiration": "0",
              "ToAccountId": 
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
              "ToOrgId": "CB",
              "ToUserId": "issuer_user_cb",
              "TokenId": "USD",
              "TokenName": "cbdc"
          }
      ],
      "encode": "JSON"
  }
}
```

`GetPendingRequest`
This method retrieves all pending requests of a specified type where the caller is assigned as an approver.

```
t.Ctx.CbdcToken.GetPendingRequest(token_id, request_type)
```

Parameters:

- `token_id: string` – The ID of the token.
- `request_type: string` – The transaction type. For example, `mint` or `burn`.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "valueJson": {
                  "AssetType": "ohold",
                  "Category": "Category value",
                  "Description": "Description value",
                  "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "HoldingId": "ohold~cbdc~USD~8e1232",
                  "NotaryAccountId":
"oaccount~2eb5f8a9bc561f8f41a4ea3be9511958cc6684ef14f2337ca396efc301b627d8",
                  "OperationId": "8e1232",
                  "OperationType": "mint",
                  "Quantity": 100,
                  "Status": "pending",
                  "TimeToExpiration": "0",
                  "ToAccountId": "",
                  "TokenId": "USD",
                  "TokenName": "cbdc"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

`GetTotalBalanceByCallerOrgId`
This method retrieves the total balance of the caller's organization.

```
t.Ctx.CbdcToken.GetTotalBalanceByCallerOrgId()
```

Parameters:

- `token_id: string` – The ID of the token.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "TotalBalance": 180
          }
      ],
      "encode": "JSON"
  }
}
```

`GetTransactionWithBlockNumber`
This method returns the details of the transaction for the specified transaction ID.

```
t.Ctx.CbdcToken.GetTransactionWithBlockNumber(token_id, transaction_id)
```

Parameters:

- `token_id: string` – The ID of the token.
- `transaction_id: string` – The ID of the transaction.

Return Value Example:

```
{
  "returnCode": "Success",
  "error": "",
  "result": {
      "payload": [
          {
              "blockNo": 152,
              "key":
"otransaction~eee2de20b4b042884da83e3b7b85d8532ad56f26a546ee25d227acce33375c1c
",
              "metadata": null,
              "txnNo": 0,
              "value": null,
              "valueJson": {
                  "Amount": 10,
                  "AssetType": "otransaction",
                  "Category": "Category value",
                  "Description": "",
                  "FromAccountBalance": 130,
                  "FromAccountId":
"oaccount~9d9806fa92aa0c4fdb34eaffac6e830181b5d47e64fbce752195e83024125ca0",
                  "FromAccountOnHoldBalance": 10,
                  "HoldingId": "ohold~cbdc~USD~hold1",
                  "NumberOfSubTransactions": 0,
                  "SubTransaction": "false",
                  "SubTransactionType": "",
                  "Timestamp": "2024-11-21T20:43:59Z",
                  "ToAccountBalance": 10,
                  "ToAccountId":
"oaccount~51e676d7182a02ea7418ef58a6d54ecfe3858ef40b4ffb3d859b320da3921594",
                  "ToAccountOnHoldBalance": 0,
                  "TokenId": "USD",
                  "TransactionId":
"otransaction~eee2de20b4b042884da83e3b7b85d8532ad56f26a546ee25d227acce33375c1c
",
                  "TransactionType": "EXECUTEHOLD"
              }
          }
      ],
      "encode": "JSON"
  }
}
```

The following SDK method is modified for the wholesale CBDC model.

`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 formed by concatenating the asset type and token ID and then creating a SHA-256 hash over a concatenation of the organization ID and user ID.

```
t.Ctx.Account.CreateAccount(org_id, user_id, token_type, application_groups,
daily_limits...)
```

Parameters:

- `orgId` – The membership service provider (MSP) ID of the user to create the account for. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `userId` – The user name or email ID of the user. The ID must begin with an alphanumeric character and can include letters, numbers, and special characters such as underscores (_), periods (.), at signs (@), and hyphens (-).
- `tokenType: TokenType` – The type of token, which must be `fungible`.
- `application_groups: string[]` – A list of application groups the user Id belongs to, which define the user's associations in the CBDC application.

- `daily_limits: DailyLimits` – A JSON object of the following type.

  ```
  {
       "max_daily_amount": 100000
       "max_daily_transactions": 10000
   }
  ```

  In the example, the `max_daily_amount` value is the maximum amount of tokens that can be transacted daily and `max_daily_transactions` value is the maximum number of transactions that can be completed daily.
- `endorsers: string[]` – An array of the peers (for example, `peer1`, `peer2`) that must endorse the transaction.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"261c3da4bb6f1724bc8f674c7b001a9b986bc9900d0508363039424926b143ed",
        "payload": {
            "AssetType": "oaccount",
            "AccountId":
"oaccount~cdc6fa5e64bc29f700f99da69f980d8cbb768c7e1a11dd17274e75651f6afafe",
            "UserId": "admin_user_cb",
            "OrgId": "CB",
            "TokenType": "fungible",
            "TokenId": "",
            "TokenName": "",
            "Balance": 0,
            "BalanceOnHold": 0,
            "BapAccountVersion": 0,
            "ApplicationGroups": [
                "CBDC_ADMINS"
            ],
            "MaxDailyAmount": 10000,
            "DailyAmount": 0,
            "MaxDailyTransactions": 100,
            "DailyTransactions": 0,
            "CurrentDate": "2024-12-09"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 246
    }
}
```

Endorsement Support in Postman Collections

Oracle Blockchain Platform Digital Assets Edition adds support for endorsement parameters to the Postman collections generated by Blockchain App Builder.

Blockchain App Builder enables you to create a Postman collection that includes example payloads for all of your chaincode controller APIs. For more information, see Generate a Postman Collection Using the CLI and Generate a Postman Collection Using Visual Studio Code.

Oracle Blockchain Platform Digital Assets Edition extends that functionality by including an additional parameter in the request payload for all setter methods. The new parameter is either `endorsers` or `sameOrgEndorser`. The `sameOrgEndorser` parameter, if `true`, indicates that transaction endorsements must be from the same organization as the requester. The `endorsers` parameter specifies a list of peers that must endorse the transaction.

The `sameOrgEndorserOptionInWrapperAPI` parameter in the `.ochain.json` file in the chaincode specifies which APIs require a `sameOrgEndorser` value. APIs that are associated with the `sameOrgEndorserOptionInWrapperAPI` parameter have the `sameOrgEndorser` parameter set to `true` in their payloads. All other APIs include the `endorsers` parameter instead of the `sameOrgEndorser` parameter.

The following snippet shows the `sameOrgEndorserOptionInWrapperAPI` parameter in the `.ochain.json` file in the wholesale CBDC chaincode package.

```
"sameOrgEndorserOptionInWrapperAPI":
["addConversionRate","addTokenAdmin","addTokenAuditor","approveBurn","approveM
int","burnTokens","createExchangePoolAccounts","deleteHistoricalTransactions",
"initializeCBDCToken","initializeExchangePoolUser","issueTokens","mintWithFund
ingExchangePool","rejectBurn","rejectMint","removeTokenAdmin","removeTokenAudi
tor","requestBurn","requestMint","updateCBDCToken","updateConversionRate"]
```

You can customize this parameter as needed. When the wrapper API is generated, the specified APIs will have the `sameOrgEndorser` parameter set to `true` in their payloads.

The following example payloads show these endorsement parameters.

`addOrgAdmin`

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "addOrgAdmin",
        "{{bc-org-id}}",
        "{{bc-user-id}}"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}},
    "endorsers": {{endorsers}}
}
```

`addTokenAdmin`

```
{
    "chaincode": "{{bc-chaincode-name}}",
    "args": [
        "addTokenAdmin",
        "{{bc-org-id}}",
        "{{bc-user-id}}"
    ],
    "timeout": {{bc-timeout}},
    "sync": {{bc-sync}},
    "sameOrgEndorser": true
}
```

Auditor Roles

The enhanced version of Blockchain App Builder includes support for auditor roles when using the extended Token Taxonomy Framework standard.

The version of Blockchain App Builder supplied with Oracle Blockchain Platform Digital Assets Edition supports two additional roles for chaincode projects that use the extended Token Taxonomy Framework standard. The new roles are `Token Auditor` and `Org Auditor`. These roles function similarly to the `Token Admin` and `Org Admin` roles, but auditor roles are limited to read-only access. Admin roles have read-write access.

The following information describes the controller methods and SDK methods that support auditor roles in both TypeScript and Go.

TypeScript Controller Methods

The following controller methods support the auditor role functions.

`addTokenAuditor`
This method adds a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
public async addTokenAuditor(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 message that includes details of the user who was added as a `Token Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"cd81f6c4c9e7c18ece357dbf5c139ef66ef2d6566be3b14de5f6d0a3fd4bb2f0",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 196
    }
}
```

`removeTokenAuditor`
This method removes a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
public async removeTokenAuditor(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 message that includes details of the user who was removed as a `Token Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"a886a6040fbc76374a3c78c89ab0ffc9f7b8391cc5239b169bf3b878cf40c67b",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 219
    }
}
```

`getTokenAuditors`
This method returns all `Token Auditors` of the chaincode. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode.

```
public async getTokenAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "auditors": [
                {
                    "org_id": "CB",
                    "user_id": "auditor_user_cb"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

addOrgAuditor

This method adds a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
public async addOrgAuditor(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 message that includes details of the user who was added as a `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"44bbad35a1478cb714e32f7cfd551897868a203520aab9cea5771d3aadc1cf03",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 198
    }
}
```

`removeOrgAuditor`
This method removes a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
public async removeOrgAuditor(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 message that includes details of the user who was removed as a `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"c3bc720461004a53b37c68d4bb264858b88d980bc093a0a3ebb62a32974fb306",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 221
    }
}
```

`getOrgAuditors`
This method returns all `Org Auditors` of the chaincode. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

```
public async getOrgAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "auditors": [
                {
                    "org_id": "FI1",
                    "user_id": "auditor_user_fi1"
                },
                {
                    "org_id": "FI2",
                    "user_id": "auditor_user_fi2"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

Go Controller Methods

The following controller methods support the auditor role functions.

`AddTokenAuditor`
This method adds a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) AddTokenAuditor(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 user who was added as a `Token Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"f0888dd52f39dfa669275cc8f35d0b47b37b8407d384493d16970fcbb377f937",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 268
    }
}
```

`RemoveTokenAuditor`

This method removes a user as a `Token Auditor` of the chaincode. This method can be called only by a `Token Admin` of the chaincode.

```
func (t *Controller) RemoveTokenAuditor(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 user who was removed as a `Token Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"2f01f48eceaf2dff175f98b96a5bdd22c949f48fc5683ce86d6141cc4892aee3",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 270
    }
}
```

`GetTokenAuditors`

This method returns all `Token Auditors` of the chaincode. This method can be called only by a `Token Admin` or `Token Auditor` of the chaincode.

```
func (t *Controller) GetTokenAuditors() (interface{}, error)
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"18eaa81b04d43db64f28287bab1cf6609e2a1d8ff84852ff73849ddb9a9dfba1",
        "payload": {
            "auditors": [
                {
                    "OrgId": "CB",
                    "UserId": "auditor_user_cb"
                }
            ]
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 269
    }
}
```

`AddOrgAuditor`

This method adds a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) AddOrgAuditor(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 user who was added as a `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"3d5ec46003c68c6208d43c82894bd6da5c0b763339cc5212e09b71d39d0d80e2",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 271
    }
}
```

RemoveOrgAuditor
This method removes a user as a `Org Auditor` of the chaincode. This method can be called only by a `Token Admin` or `Org Admin` of the chaincode.

```
func (t *Controller) RemoveOrgAuditor(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 user who was removed as a `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"713a120641afbc4dfaeac73b82c9fd51df6fcfd7d4d9a82553d3c487bf11f530",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 273
    }
}
```

GetOrgAuditors
This method returns all `Org Auditors` of the chaincode. This method can be called only by a `Token Admin`, `Token Auditor`, `Org Admin`, or `Org Auditor`.

```
func (t *Controller) GetOrgAuditors() (interface{}, error)
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"07656bf434616d7a3d7fe4fb81dc80c8cc608991648adfd9f6f2f2b9f6ddf468",
        "payload": {
            "auditors": [
                {
                    "OrgId": "CB",
                    "UserId": "cb"
                },
                {
                    "OrgId": "CB",
                    "UserId": "issuer_user_cb"
                }
            ]
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 272
    }
}
```

TypeScript SDK Methods

The following SDK methods support the auditor role functions.

`addTokenAuditor`
This method adds a user as a `Token Auditor` of the chaincode.

```
this.Ctx.Admin.addTokenAuditor(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 Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"cd81f6c4c9e7c18ece357dbf5c139ef66ef2d6566be3b14de5f6d0a3fd4bb2f0",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 196
    }
}
```

`removeTokenAuditor`
This method removes a user as a `Token Auditor` of the chaincode.

```
this.Ctx.Admin.removeTokenAuditor(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 Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"a886a6040fbc76374a3c78c89ab0ffc9f7b8391cc5239b169bf3b878cf40c67b",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 219
    }
}
```

`getTokenAuditors`
This method returns all `Token Auditors` of the chaincode.

```
this.Ctx.Admin.getAllTokenAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "auditors": [
                {
                    "org_id": "CB",
                    "user_id": "auditor_user_cb"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

addOrgAuditor
This method adds a user as a `Org Auditor` of the chaincode.

```
this.Ctx.Admin.addOrgAuditor(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 `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"44bbad35a1478cb714e32f7cfd551897868a203520aab9cea5771d3aadc1cf03",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20009",
        "blockNumber": 198
    }
}
```

removeOrgAuditor
This method removes a user as a `Org Auditor` of the chaincode.

```
this.Ctx.Admin.removeOrgAuditor(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 `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid": 
"c3bc720461004a53b37c68d4bb264858b88d980bc093a0a3ebb62a32974fb306",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id: 
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 221
    }
}
```

`getOrgAuditors`
This method returns all `Org Auditors` of the chaincode.

```
this.Ctx.Admin.getAllOrgAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "payload": {
            "auditors": [
                {
                    "org_id": "FI1",
                    "user_id": "auditor_user_fi1"
                },
                {
                    "org_id": "FI2",
                    "user_id": "auditor_user_fi2"
                }
            ]
        },
        "encode": "JSON"
    }
}
```

Go SDK Methods

The following controller methods support the auditor role functions.

`AddTokenAuditor`
This method adds a user as a `Token Auditor` of the chaincode.

```
t.Ctx.Admin.AddTokenAuditor(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 Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"f0888dd52f39dfa669275cc8f35d0b47b37b8407d384493d16970fcbb377f937",
        "payload": {
            "msg": "Successfully added Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 268
    }
}
```

`RemoveTokenAuditor`
This method removes a user as a `Token Auditor` of the chaincode.

```
t.Ctx.Admin.RemoveTokenAuditor(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 Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"2f01f48eceaf2dff175f98b96a5bdd22c949f48fc5683ce86d6141cc4892aee3",
        "payload": {
            "msg": "Successfully removed Token Auditor (Org_Id: CB, User_Id:
cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 270
    }
}
```

`GetTokenAuditors`
This method returns all `Token Auditors` of the chaincode.

```
t.Ctx.Admin.GetAllTokenAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"18eaa81b04d43db64f28287bab1cf6609e2a1d8ff84852ff73849ddb9a9dfba1",
        "payload": {
            "auditors": [
                {
                    "OrgId": "CB",
                    "UserId": "auditor_user_cb"
                }
            ]
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 269
    }
}
```

`AddOrgAuditor`
This method adds a user as a `Org Auditor` of the chaincode.

```
t.Ctx.Admin.AddOrgAuditor(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 `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"3d5ec46003c68c6208d43c82894bd6da5c0b763339cc5212e09b71d39d0d80e2",
        "payload": {
            "msg": "Successfully added Org Auditor (Org_Id: CB, User_Id: cb)"
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 271
    }
}
```

RemoveOrgAuditor
This method removes a user as a `Org Auditor` of the chaincode.

```
t.Ctx.Admin.RemoveOrgAuditor(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 `Org Auditor` of the chaincode.

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"713a120641afbc4dfaeac73b82c9fd51df6fcfd7d4d9a82553d3c487bf11f530",
        "payload": {
            "msg": "Successfully removed Org Auditor (Org_Id: CB, User_Id:
cb)"
```

```
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 273
    }
}
```

GetOrgAuditors
This method returns all `Org Auditors` of the chaincode.

```
t.Ctx.Admin.GetAllOrgAuditors()
```

Return Value Example:

```
{
    "returnCode": "Success",
    "error": "",
    "result": {
        "txid":
"07656bf434616d7a3d7fe4fb81dc80c8cc608991648adfd9f6f2f2b9f6ddf468",
        "payload": {
            "auditors": [
                {
                    "OrgId": "CB",
                    "UserId": "cb"
                },
                {
                    "OrgId": "CB",
                    "UserId": "issuer_user_cb"
                }
            ]
        },
        "encode": "JSON",
        "sourceURL": "cb-oabcs1-bom.blockchain.ocp.example.com:20010",
        "blockNumber": 272
    }
}
```

Method Level Access Controls

Auditor roles have read-only access to methods that use the extended Token Taxonomy Framework standard.

The following table shows how the auditor roles affect access to methods in the extended Token Taxonomy Framework standard.

Method name	Roles that can access the method
`isTokenAdmin`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getAllAdmins`	`Token Admin`, `Token Auditor`

Method name	Roles that can access the method
`getOrgAdmins`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getAllTokens`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getTokenById`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getTokenDecimals`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getTokensByName`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`isInRole`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization, `Account Owner`
`getTotalMintedTokens`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getNetTokens`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getTokenHistory`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getAccountsByRole`	`Token Admin`, `Token Auditor`
`getOrgAccountsByRole`	`Token Admin`, `Token Auditor`, `Org Admin` of any organization, `Org Auditor` of any organization
`getUsersByRole`	`Token Admin`, `Token Auditor`
`getOrgUsersByRole`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization, `Account Owner`
`getTransactionById`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization, transaction participant (sender, recipient, notary)
`getAllAccounts`	`Token Admin`, `Token Auditor`
`getAllOrgAccounts`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization
`getAccountsByUser`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization, multiple account owner
`getUserByAccountId`	`Token Admin`, `Token Auditor`, `Org Admin` of the particular organization, `Org Auditor` of the particular organization

Method name	Roles that can access the method
getAccount	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getAccountTransactionHistory	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getAccountTransactionHistoryWithFilters	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getSubTransactionsById	Token Admin, Token Auditor, transaction invoker
getSubTransactionsById	Token Admin, Token Auditor, transaction invoker
getAccountBalance	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getAccountOnHoldBalance	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getOnHoldIds	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getOnHoldDetailsWithOperationId	Token Admin, Token Auditor, hold transaction participant (sender, recipient, notary)
getOnHoldBalanceWithOperationId	Token Admin, Token Auditor, hold transaction participant (sender, recipient, notary)
getConversionHistory	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getAccountStatus	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getAccountStatusHistory	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, Account Owner
getConversionRate	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, any account owner
getConversionRateHistory	Token Admin, Token Auditor, Org Admin of the particular organization, Org Auditor of the particular organization, any account owner
getExchangePoolUser	Token Admin, Token Auditor