BSN-DDC
Solidity Contract Design

V1.5

Red Date Technology

December 2021

Contents

1. Purpose

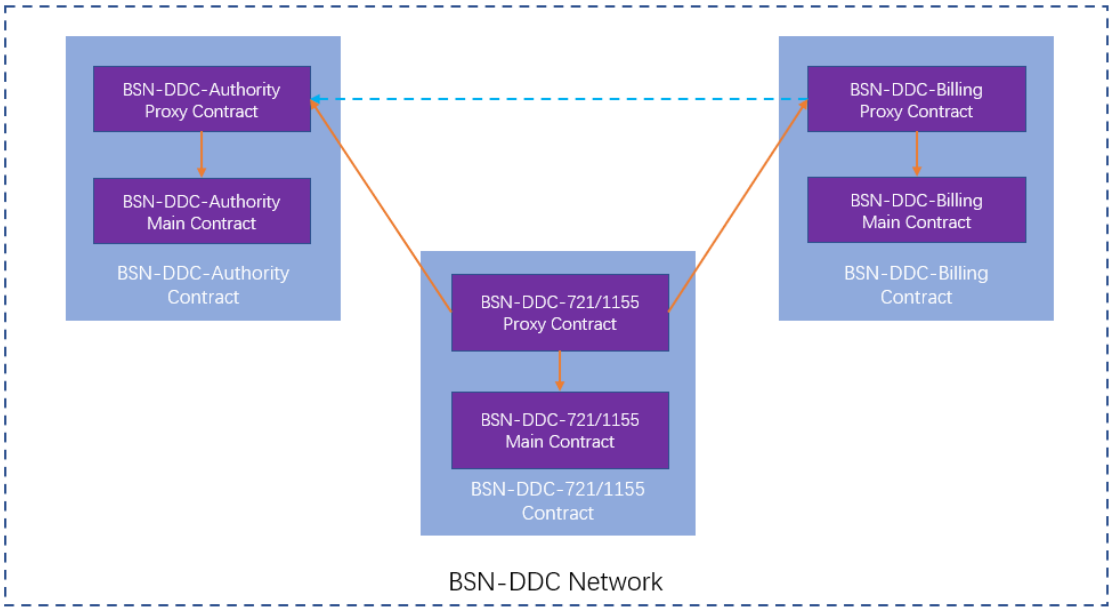
This document was created to help project team members and all Open Permissioned Blockchain (OPB) framework providers have a comprehensive and detailed understanding of the overall design of a BSN-DDC contract, and provide guidance for the development, test, validation, delivery and other parts of the project.

2.Requirement Document

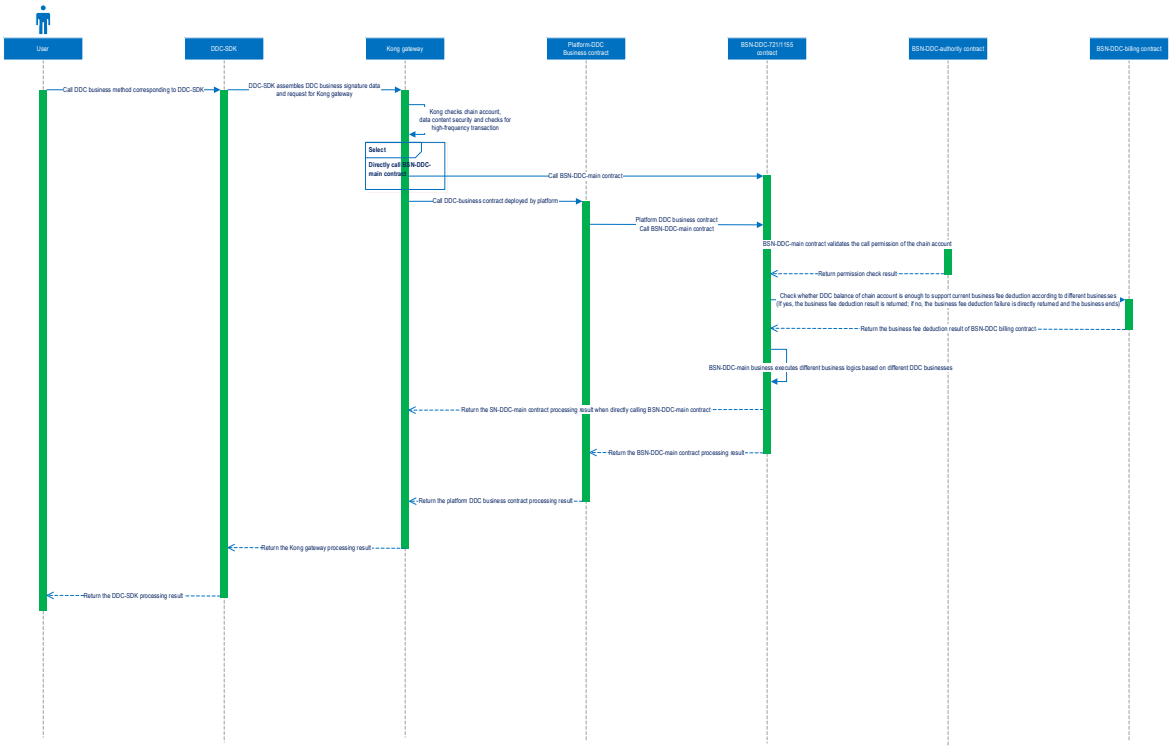
The requirement document refers to the BSN-DDC Requirements Specification V1.1.docx.

3.Overall Design

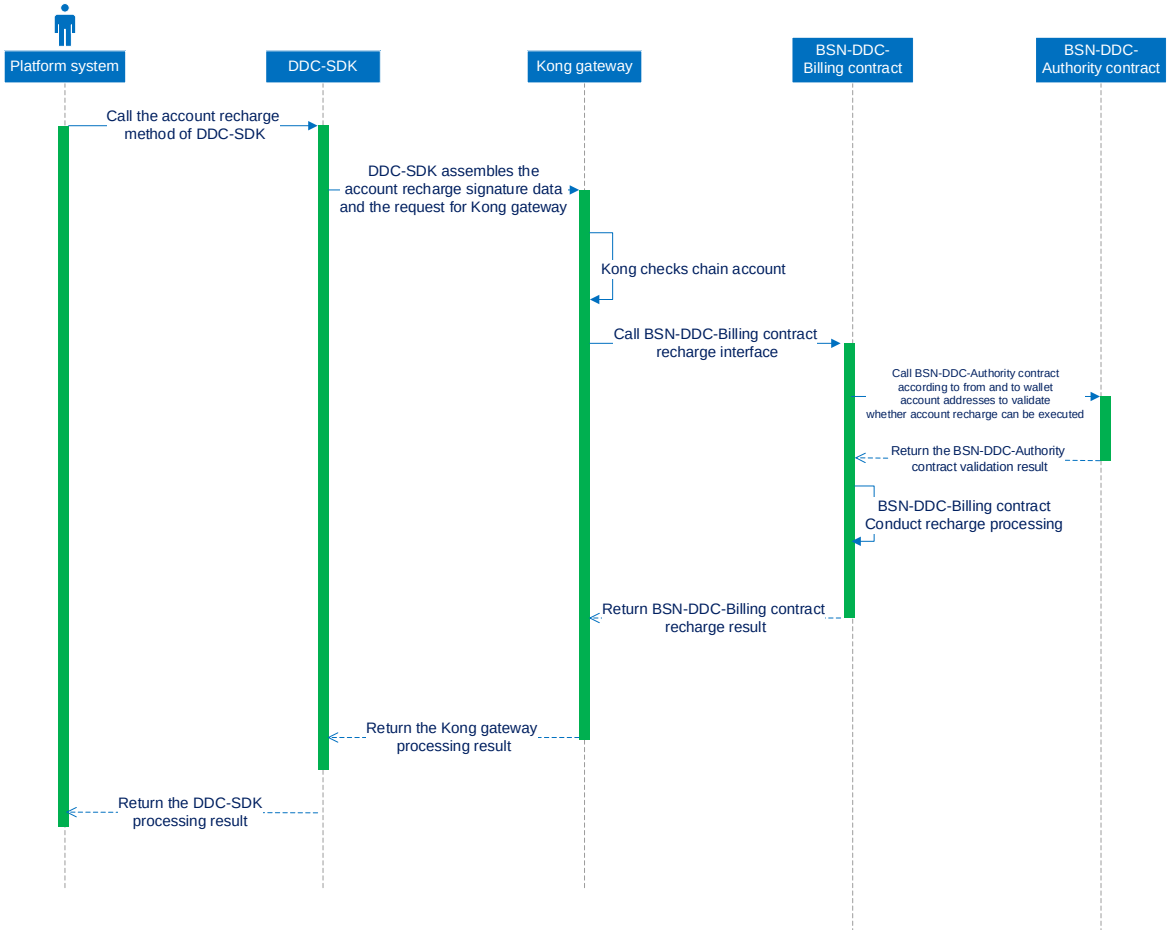
3.1 Overall Contract Structure



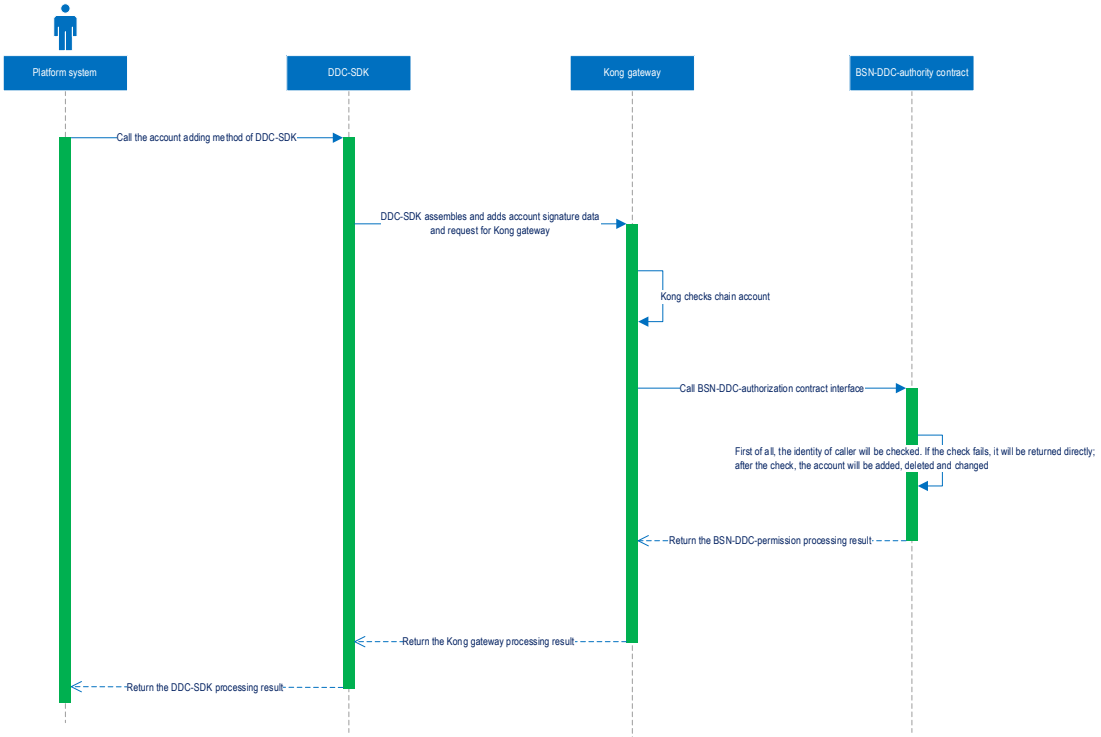
3.2 DDC Business Transaction Timing Diagram



3.3 Billing Recharge Transaction Timing Diagram



3.4 Account Management Transaction Timing Diagram



3.5 Security Design Description

Currently, the overall design of a BSN-DDC contract adopts a 2-layer design approach, including a main contract and a proxy contract. The main contract can only be called by the corresponding proxy contract. For example, a BSN-DDC-Billing contract can only be called by a BSN-DDC-Billing proxy contract.
Note: Business contracts need to define corresponding interfaces, and the business processing is carried out in the main contract.

3.6 Description of Contract Update Design

BSN-DDC main contracts can be upgraded via UUPS (EIP-1822: Universal Upgradeable Proxy Standard). Every BSN-DDC business has its proxy contract, whose owner is the BSN-DDC Operator. Platform Owners or their business contracts can call a proxy contract to access the corresponding main contract.

- The update of a main contract as follows:

1. Inherit UUPS Upgradeable, which is an upgradeability mechanism for the proxy design of UUPS.
2. Add “initialize()” function to initialize a main contract when calling and deploying contracts.

- The deployment of a main contract is as follows:
  1. Deploy the main contract.
  2. Deploy the proxy contract. When deploying the contract, construct parameters, write the address of the main contract, sign the initialize() function to map the proxy contract and the main contract, and initialize both contracts.
- The upgrade process of a main contract is as follows:
  1. Deploy a new main contract.
  2. Call the upgradeTo function in the current contract and upload the new main contract address to update the main contract by binding the mapping relationship with the new main contract.

4 Contract Design

4.1 BSN-DDC-Billing Contract

Service fee defines the measuring unit name of the BSN-DDC service fee.

4.1.1 Function Description

The BSN-DDC-Billing contract is used to unify the management of the on-chain accounts of all parties involved in the DDC business, including the definition of billing rules and the service fees deducted by various types of accounts when calling BSN-DDC official contracts according to the billing rules.

On-chain accounts include the following:

1. Operator: Operator account in the BSN-DDC-Billing contract;

2. Platform Owner: Platform Owner account in the BSN-DDC-Billing contract.

3. End User: End-User account in the BSN-DDC-Billing contract with each account belonging to one Platform Owner.

4.1.2 Data structure

- DDC billing rules

No.	Field name	Field	Type	Remarks
1.	BSN-DDC-721/1155 proxy contract address	key	address	BSN-DDC-721/1155 proxy contract address
2.	Billing rules set	value	FuncFee	The cost and DDC deletion state of each function of the BSN-DDC main contract
FuncFee				
1.	Service fee of sig function	funcfee	mapping(bytes4 => uint32)	The sig function and its corresponding service fee of the BSN-DDC-721/1155 proxy contract
2.	BSN-DDC-721/1155 proxy contract State	used	bool	Deletion status of BSN-DDC-721/1155 proxy contract

- User account

No.	Field name	Field	Type	Remarks
1.	Account address	key	address	User account address
2.	Account balance	value	uint32	User account balance

4.1.3 API Definition

4.1.3.1 Recharge DDC Service Fee

Operators and Platform Owners can recharge the DDC service fees of their subordinate accounts through this API.

- Input parameters: receiver account, service fee to be recharged;
- Output parameter:
- Function name: recharge;
- Function example: recharge(address to,uint256 amount);

- Event: Recharge(address indexed from, address indexed to,uint256 amount);
- Core logic
  - Check whether the recharged service fee is 0. If it is, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the transferor account and receiver account are the same. If it is, return a prompt message; Check whether the transferor account is active. If not, return a prompt message;
  - Check whether the receiver account is active. If not, return a prompt message;
  - Check whether the following conditions are met. If none of them are met, then return a prompt message;
    - Check whether the transferor account type is Operator, who will recharge the accounts of Platform Owners or end users.
    - Or check whether the transferor’s account DID and the leader DID of the receiver account are the same. In this circumstance, Operator will recharge Platform Owners’ accounts or Platform Owners will recharge end users accounts.
    - Or check whether the leader DIDs of transferors are the same as those of the receivers. In this circumstance, account DIDs of transferors and receivers accounts are the same and the receiver should not be an end user, and Platform Owners’ accounts will recharge each other.
  - Check whether transferor’s account balance is greater than or equal to the recharged service fee. If not, return a prompt message;
  - Save the recharge results after all the checks and trigger the event;
- Business rules

- The Operator can recharge the accounts of Platform Owners and end users;
- Platform Owners can recharge its own platform users' accounts, but they cannot recharge other platforms and end users' accounts;
- End users cannot call this function to recharge any account;

➢ Business scenarios

- When an Operator is recharging a Platform Owner's account, he/she will call the recharge API to top up Platform Owner's account;
- End users recharge their accounts in the platform, Platform Owners will call the recharge API to top up end users accounts.

4.1.3.2 Batch Recharge DDC Service Fee

Operators and Platform Owners can recharge the DDC service fees of their subordinate accounts in a batch through this API.

➢ Input parameters: receiver account set, set of service fee to be recharged;

➢ Output parameter:

➢ Function name: rechargeBatch;

➢ Function example: rechargeBatch(address[] memory toList,uint256[] memory amounts);

➢ Event: RechargeBatch(address indexed from, address[] indexed toList,uint256[] amounts);

➢ Core logic

- Check if the receiver account set length is equal to the service fee set length. If not, return a prompt message;
- Check whether the transferor account is active. If not, return a prompt message;
- Batch loop to process chain account service fee recharge, the processing logic is as follows:

1. Check whether the amount of the service fee to be recharged is equal to 0 according to the index. If it is, return a prompt message;

2. Check whether the receiver account address is a “0 address” according to the index. If it is, return a prompt message;

3. Check whether the receiver account is the same as the transferor account according to the index. If it is, return a prompt message;

4. Check if the receiver account status is active according to the index. If not, return a prompt message;

5. Check whether the following conditions are met. If none of them are met, then return a prompt message:

- Check whether the transferor account type is Operator, who will recharge the accounts of Platform Owners or end users;
- Or check whether the transferor’s account DID and the leader DID of the receiver account are the same. In this circumstance, Operator will recharge Platform Owners’ accounts or Platform Owners will recharge end users accounts;
- Or check whether the leader DIDs of transferors are the same as those of the receivers. In this circumstance, account DIDs of transferors and receivers accounts are the same and the receiver should not be an end user, and Platform Owners’ accounts will recharge each other;

6. Check whether the transferor account balance is greater than or equal to the amount of the service to be recharged according to the index. If not, return a prompt message;

7. Save the recharge result after all checks are passed.

- Trigger the event.

4.1.3.3 Recharge Operator Account

When the total service fee in an Operator account is insufficient to pay for the

current service, the Operator can recharge the total service fee.

- Input parameter: service fee to recharged;
- Output parameter:
- Caller:
- Function name: selfRecharge;
- Function example: selfRecharge(uint amount);
- Event: Recharge(address indexed from, address indexed to,uint256 amount);
- Core logic:
  - Check whether the recharge amount of the service fee is greater than 0. If not, return a prompt message;
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the account type of the caller is Operator. if not, return a prompt message;
  - Recharge the account of Operator after all the checks and trigger the event (“from” is an empty address while “to” is the caller address);
- Business rules: It is only allowed to recharge the total service fee to the Operator account;

4.1.3.4 DDC Service Fee Deduction

Call this API in the BSN-BSN-DDC-721/1155 proxy contract and upload user address and contract identification (sig) to transfer the user account balance to the operation account according to the billing rules;

- Input parameters: transferor account, function signature;
- Output parameter:
- Function name: pay;
- Function example: pay(address payer, bytes4 sig) returns (bool success);

- Event: Pay(address indexed payer, address indexed payee,bytes4 sig,uint32 amount,uint256 ddcId);
- Core logic:
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the billing rules of BSN-DDC-721/1155 proxy contract exists or available. If not, return a prompt message;
  - Check the service fee of BSN-DDC-721/1155 proxy contract and function sig is greater than 0. If not, trigger the event;
  - Check whether the transferor account balance is greater than or equal to the deducted service fee. If not, return a prompt message;
  - After all checks are passed, deduct the service fee from transferor account balance and recharge the account of BSN-DDC-721/1155 proxy contract, and trigger the event;
- Business rules: This function can only be called in the BSN-DDC-721/1155 proxy contract;

4.1.3.5 DDC Contract Settlement

The operation account can call this function to initiate settlement for the account of the authorized BSN-DDC-721/1155 proxy contract;

- Input parameter: settlement account, settlement amount;
- Output parameter:
- Function name: settlement;
- Function example: settlement(address ddcAddr,uint256 amount);
- Event: Settlement(address indexed accAddr, address indexed ddcAddr,uint256 amount);
- Core logic:
  - Check whether the settlement amount is greater than 0. If not, return a prompt message;

- Check whether settlement account is a BSN-DDC-721/1155 proxy contract that exists in the billing rules. If not, return a prompt message;
- Check whether the caller account is available. If not, return a prompt message;
- Check whether the account type of caller is Operator. If not, return a prompt message;
- Check whether the amount of settlement account balance is greater than or equal to the settlement amount. If not, return a prompt message;
- Proceed to make the settlement after all checks are passed and trigger the event;

4.1.3.6 Query Chain Account Balance

An Operator, Platform Owner, or end user can query the DDC service fee balance of a user via this API.

- Input parameter: account address;
- Output parameters: account balance;
- Function name: balanceOf;
- Function example: balanceOf(address accAddr) view returns (uint256 balance);
- Core logic:
  - Check whether the account address is “0 address”. If it is, return a prompt message;
  - Return the query results after all checks are passed;

4.1.3.7 Batch Query Chain Account Balance

An Operator, Platform Owner, or end user can query the DDC service fee balance of users in a batch via this API.

- Input parameter: account address set;

- Output parameters: account balance set;
- Function name: balanceOfBatch;
- Function example: balanceOfBatch(address[] memory accAddrs) view returns (uint256[] memory);
- Core logic:
  - Batch loop to query the chain account balance one by one. Check whether the chain account address is “0 address” according to the index. If it is, return the prompt message; otherwise the chain account balance corresponding to each query will be added to the set list of chain account balance, and return the result;

4.1.3.8 Set the Billing Rules of DDC Service Fee

The Operator account can call this API to authorize the setting of the DDC contract billing rules. The original billing rules will be overwritten if they have been set. This means that this contract is the BSN-DDC-Authority contract which can call the BSN-DDC-Billing contract to deduct fees from the caller account.

- Input parameters: BSN-DDC-721/1155 proxy contract address, contract function sig, service fee amount;
- Output parameter:
- Function name: setFee;
- Function example: setFee(address ddcAddr, byte4 sig,uint32 amount);
- Event: SetFee(address ddcAddr, byte4 sig, uint amount);
- Contract logic:
  - Check whether the BSN-DDC-721/1155 proxy contract address is “0 address”. If it is, return a prompt message;
  - Check whether the caller account is available. If it is, return a prompt message;
  - Check whether account type of the caller is Operator. If not, return a prompt message;

- Save billing rules data after all checks are passed and trigger the event;

4.1.3.9 Delete the Billing Rules of DDC Service Fee

An Operator can call this API to delete the set DDC contract billing rules. They can still authorize a BSN-DDC-721/1155 proxy contract even if all the billing rules are deleted.

- Input parameter: BSN-DDC-721/1155 main contract address;
- Output parameter:
- Function name: delFee;
- Function example: delFee(address ddcAddr, bytes4 sig);
- Event: DelFee(address ddcAddr, bytes4 sig);
- Contract logic:
  - Check whether the BSN-DDC-721/1155 proxy contract address is “0 address”. If it is, return a prompt message;
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the account type of the caller is Operator. If not, return a prompt message;
  - Delete the billing rules data of the DDC contract after all checks are passed and trigger the event;

4.1.3.10 Delete BSN-DDC-721/1155 Contract Authorization

If an Operator calls this API to delete the DDC contract, it means that the DDC contract will no longer be managed through authorization. The permission for this contract to call the billing contract for fee deduction and the billing rules that have been set in the billing contract will also be deleted.

- Input parameter: DDC main contract address;
- Output parameter:

- Function name: delDDC;
- Function example: delDDC (address ddcAddr);
- Event: DelDDC(address ddcAddr);
- Contract logic:
  - Check whether the BSN-DDC-721/1155 proxy contract address is “0 address”. If it is, return a prompt message;
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the account type of the caller is Operator. If not, return a prompt message;
  - Delete the BSN-DDC-721/1155 proxy contract and all data corresponding to the billing rules of that contract after all checks are passed and trigger the event;

4.1.3.11 Query Billing Rules of DDC Service Fee

An Operator, Platform Owner, or end-user can query the service fee to call each API in the DDC contract.

- Input parameter: BSN-DDC-721/1155 proxy contract address;
- Output parameter: service fee of function Sig for each BSN-DDC-721/1155 proxy contract;
- Function name: queryFee;
- Function example: queryFee(address ddcAddr, bytes4 sig) view returns (uint amount);
- Core logic:
  - Check whether the BSN-DDC-721/1155 proxy contract address is “0 address”.
  - Check whether the billing rules of BSN-DDC-721/1155 proxy contract is available. If not, return a prompt message;
  - Return the query results after all checks are passed;

4.1.3.12 Query Total Service Fee

An Operator can query the total on-chain service fee by the billing contract.

- Input parameter:
- Output parameter: total service fee;
- Function name: totalSupply;
- Function example: totalSupply() view returns (uint256 totalSupply);
- Core logic:
  - Query the total on-chain service fee;

4.2 BSN-DDC-721 Main Contract

4.2.1 Function Description

The BSN-DDC-721 main contract is used to provide chain accounts with a set of APIs, including BSN-DDC generation, authorization, query authorization, transfer, freezing, unfreezing and deduction.

4.2.2 Data Structure

- DDC storage structure

No.	Field name	Field	Type	Remarks
1.	DDC name	_name	String	
2.	DDC symbol	_symbol	string	
3.	DDC URI set	_ddcURIs	mapping<uint256=>string>	Key: DDC ID Value: URI
4.	DDC owner address mapping set	_owners	mapping(uint256 => address)	Key: DDC ID Value: owner address
5.	DDC balance mapping set	_balances	mapping(address => uint256)	Key: owner address Value: DDC number
6.	DDC authorizer mapping set	_ddcApprovals	mapping(uint256 => address)	Key: DDC ID Value: authorized person address
7.	DDC operator	_operatorA	mapping(address	Key: DDC

	mapping set	pprovals	=> mapping(address => bool))	owner address Value: authorized person address and option whether to authorize
8.	DDC blacklist	_blacklist	mapping<uint256=>bool>	Key: DDC ID Value: DDC state
9.	The latest DDC id	_lastDDCId	uint256	

4.2.3 API Definition

4.2.3.1 Generation

A Platform Owner or an end-user can call this API to mint a DDC.

- Input parameters: receiver account, URI;
- Output parameter:
- Function name: mint;
- Function example: mint(address to, string memory ddcURI);
- Event: Transfer(operator, address(0), to, ddcId);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
  - Generate the DDC ID and check whether the DDC ID already exists. If it does, return a prompt message;

- Call the billing contract to pay for the DDC service fee after all checks are passed and save the generated DDC data and trigger the event;

4.2.3.2 Batch Generation

A Platform Owner or an end-user can call this API to mint DDCs in a batch.

- Input parameters: receiver account, URI;
- Output parameter:
- Function name: mintBatch;
- Function example: mintBatch(address to,string[] memory ddcURIs);
- Event: TransferBatch(operator, address(0), to, ddcIds);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
  - Batch loop to process DDC generation one by one, and check whether each generated DDC ID exists according to the index. If it does, return the prompt message, otherwise call the billing contract to pay the DDC service fee one by one and save the generated DDC data;
  - Trigger the event;

4.2.3.3 Safe Generation

A Platform Owner or an end-user can call this API to securely generate a DDC.

- Input parameters: receiver account, URI, additional data;
- Output parameter:
- Function name: safeMint;
- Function example: safeMint(address to, string memory ddcURI, bytes memory data);
- Event: Transfer(operator, address(0), to, ddcId);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
  - Generate the DDC ID and check whether the DDC ID already exists. If it does, return a prompt message;
  - Call the billing contract to pay for the DDC service fee after all checks are passed and save the generated DDC data;
  - Trigger the event. Check whether the receiver account receives DDC if the receiver account is a contract;

4.2.3.4 Batch Safe Generation

A Platform Owner or an end-user can call this API to securely generate DDCs in

a batch.

- Input parameters: receiver account, URI, additional data;
- Output parameter:
- Function name: safeMintBatch;
- Function example: safeMintBatch(address to, string[] memory ddcURIs, bytes memory data);
- Event: TransferBatch(operator, address(0), to, ddcIds);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
  - Batch loop to process DDC safe generation one by one, and check whether each generated DDC ID exists according to the index. If it does, return the prompt message, otherwise call the billing contract to pay the DDC service fee one by one and save the generated DDC data;
  - Trigger the event. Check whether the receiver account receives DDC if the receiver account is a contract;

4.2.3.5 DDC Authorization

A DDC owner can call this API to authorize a DDC.

- Input parameters: authorized account, DDC ID;
- Output parameter:

- Function name: approve;
- Function example: approve(address to,uint256 ddcId);
- Event: Approval(operator, to, ddcId);
- Core logic:
  - Check whether the caller is available. If not, return a prompt message;
  - Check whether the caller has call permission. If not, return a prompt message;
  - Check whether the authorized account address is “0 address”. If it is, return a prompt message;
  - Check whether the authorized account is available. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDC ID is frozen. If it is, return a prompt message;
  - Check whether the caller account and the authorized account belong to the same platform. If not, return a prompt message;
  - Check whether the DDC owner account and the authorized account are the same account. If they are, return a prompt message;
  - Check whether the DDC owner account and caller account are the same account, or check whether the DDC caller account is authorized by the DDC owner account. If not, return a prompt message;
  - Call the billing contract to pay for DDC service fee after all checks are passed and save the DDC authorization data and trigger the event;
  - Note: Only DDCs of the current owner can be authorized. The authorization will be invalid after transfer or safe transfer;

4.2.3.6 Batch DDC Authorization

A DDC owner can call this API to authorize DDCs in a batch.

- Input parameters: authorized account, DDC ID set;
- Output parameter:

- Function name: approveBatch;
- Function example: approveBatch(address to,uint256[] memory ddcIds);
- Event: ApprovalBatch(operator, to, ddcIds);
- Core logic:
  - Check whether the caller is available. If not, return a prompt message;
  - Check whether the caller has call permission. If not, return a prompt message;
  - Check whether the authorized account address is “0 address”. If it is, return a prompt message;
  - Check whether the authorized account is available. If not, return a prompt message;
  - Batch loop to process DDC authorization one by one, the processing logic is as follows:

    1. Check whether the DDC ID exists according to the index. If not, return a prompt message.

    2. Check whether the DDC ID is frozen according to the index. If it is, return a prompt message.

    3. Check whether the owner of the DDC is the same as the authorized account according to the index. If it is, return the prompt message.

    4. Check whether the owner and the caller account corresponding to the DDC belong to the same account, or check whether the owner account corresponding to the DDC is authorized to the caller account according to the check index. If not, return the prompt message.

    5. After all checks are passed, call the billing contract to pay the DDC service fee and save the DDC authorization data.
  - Trigger the event;
  - Note: Only DDCs of the current owner can be authorized. The authorization will be invalid after transfer or safe transfer.

4.2.3.7 Query DDC Authorization

An Operator, Platform Owner, or end-user can call this API to query the DDC authorization.

- Input parameter: DDC ID;
- Output parameter: authorized account
- Function name: getApproved;
- Function example: getApproved(uint256 ddcId) view returns (address);
- Core logic:
  - Check whether the DDC ID exists. If not, return a prompt message;
  - The query result will be returned after all checks are passed.

4.2.3.8 Account Authorization

A DDC owner can call this API to authorize the account.

- Input parameters: authorized account, authorization identifier
- Output parameter:
- Function name: setApprovalForAll;
- Function example: setApprovalForAll(address operator, bool approved);
- Event: ApprovalForAll(owner, operator, approved);
- Core logic:
  - Check whether the caller is available. If not, return a prompt message;
  - Check whether the caller has call permission. If not, return a prompt message;
  - Check whether the authorized account address is “0 address”. If it is, return a prompt message;
  - Check whether the authorized account is available. If not, return a prompt message;
  - Check whether the caller account and the authorized account belong to the same platform. If not, return a prompt message;

- Check whether the DDC owner account and the authorized account are the same account. If they are, return a prompt message;
- Pay for the DDC service fee after all checks are passed, call the billing contract to pay DDC service fee and save the DDC authorization data, and trigger the event;
- Note: Only DDCs of the current owner can be authorized. The authorization will be invalid after transfer.

4.2.3.9 Verify Account Authorization

An Operator, Platform Owner, or end-user can call this API to verify the account authorization.

- Input parameters: owner account, authorized account
- Output parameter: bool result;
- Function name: isApprovedForAll;
- Function example: isApprovedForAll(address owner, address operator) view returns (bool);
- Core logic:
  - Check whether the owner account or authorized account is 0. If it is, return a prompt message;
  - Query whether the authorized account is authorized by the owner account and return the validation result.

4.2.3.10 Safe Transfer

A DDC owner or an authorized account can call this API to securely transfer DDCs.

- Input parameters: owner account, receiver account, DDC ID, additional data;
- Output parameter:

- Function name: safeTransferFrom;
- Function example: safeTransferFrom(address from, address to, uint256 ddcId, bytes memory data);
- Event: Transfer(operator, from, to, ddcId);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the owner account is available. If not, return a prompt message;
  - Check whether the owner account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the DDC owner account and receiver account belong to the same platform or authorized by a different platform. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDCID is frozen. If it is, return a prompt message;
  - Check whether the DDC owner account and caller account are the same account, or check whether the authorized account and caller account are the same account, or check whether the DDC caller account is authorized by the owner account. If not, return a prompt message;
  - Check whether the DDC owner account and the input owner account are the same account. If not, return a prompt message;

- Call the billing contract to pay for DDC service fee, save the DDC safe transfer data and remove the DDC ID authorization list;
- Trigger the event. Check whether the receiver account can receive DDC if it is a contract.

4.2.3.11 Batch Safe Transfer

A DDC owner or an authorized account can call this API to securely transfer DDCs in a batch.

- Input parameters: owner account, receiver account, DDC ID, additional data;
- Output parameter:
- Function name: safeTransferFrom;
- Function example: safeBatchTransferFrom(address from, address to,uint256[] memory ddcIds, bytes memory data);
- Event: Transfer(operator, from, to, ddcIds);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the owner account is available. If not, return a prompt message;
  - Check whether the owner account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;

- Check whether the DDC owner account and receiver account belong to the same platform or authorized by a different platform. If not, return a prompt message;
- Batch loop to process DDC safe transfer one by one, the processing logic is as follows:

  1. Check whether the DDC ID exists according to the index. If not, return a prompt message.

  2. Check whether the DDC ID is frozen according to the index. If it is, return a prompt message.

  3. Check whether the owner of the DDC is the same as the authorized account according to the index. If it is, return the prompt message.

  4. Check whether the owner and the caller account corresponding to the DDC belong to the same account, or check whether the owner account corresponding to the DDC is authorized to the caller account according to the check index. If not, return the prompt message.

  5. After all checks are passed, call the billing contract to pay the DDC service fee and save the DDC safe transfer data.
- Trigger the event. Check whether the receiver account can receive DDC if it is a contract.

4.2.3.12 Transfer

A DDC owner or authorized account can call this API to transfer DDCs.

- Input parameters: owner account, receiver account, DDC ID;
- Output parameter:
- Function name: transferFrom;
- Function example: transferFrom(address from, address to,uint256 ddcId);
- Event: Transfer(operator, from, to, ddcId);
- Core logic:

- Check whether the caller account is available. If not, return a prompt message;
- Check whether the caller account has the permission. If not, return a prompt message;
- Check whether the owner account is available. If not, return a prompt message;
- Check whether the owner account address is “0 address”. If it is, return a prompt message;
- Check whether the receiver account is available. If not, return a prompt message;
- Check whether the receiver account address is “0 address”. If it is, return a prompt message;
- Check whether the DDC owner account and receiver account belong to the same platform or authorized by a different platform. If not, return a prompt message;
- Check whether the DDC ID exists. If not, return a prompt message;
- Check whether the DDCID is frozen. If it is, return a prompt message;
- Check whether the DDC owner account and caller account are the same account, or check whether the authorized account and caller account are the same account, or check whether the DDC caller account is authorized by the owner account. If not, return a prompt message;
- Check whether the DDC owner account and the owner account parameters belong to the same account. If not, return a prompt message;
- Call the billing contract to pay for DDC service fee, save the DDC transfer data and remove the DDC ID authorization list, and trigger the event;

4.2.3.13 Batch Transfer

A DDC owner or authorized account can call this API to transfer DDCs in a batch.

- Input parameters: owner account, receiver account, DDC ID set;

- Output parameter:
- Function name: batchTransferFrom;
- Function example: batchTransferFrom(address from, address to,uint256[] memory ddcIds);
- Event: TransferBatch(operator, from, to, ddcIds);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the owner account is available. If not, return a prompt message;
  - Check whether the owner account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the DDC owner account and receiver account belong to the same platform or authorized by a different platform. If not, return a prompt message;
  - Batch loop to process DDC transfer one by one, the processing logic is as follows:

    1. Check whether the DDC ID exists according to the index. If not, return a prompt message.

    2. Check whether the DDC ID is frozen according to the index. If it is, return a prompt message.

    3. Check whether the owner of the DDC is the same as the authorized account according to the index. If it is, return the prompt message.

4. Check whether the owner and the caller account corresponding to the DDC belong to the same account, or check whether the owner account corresponding to the DDC is authorized to the caller account according to the check index. If not, return the prompt message.

5. After all checks are passed, call the billing contract to pay the DDC service fee and save the DDC transfer data.

- Trigger the event.

4.2.3.14 Freezing

An Operator can call this API to freeze DDCs.

- Input parameter: DDC ID;
- Output parameter:
- Function name: freeze;
- Function example: freeze(uint256 ddcId);
- Event: EnterBlacklist(sender, ddcId);
- Core logic:
  - Check whether the caller account state is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the caller account is Operator. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDCID is frozen. If it is, return a prompt message;
  - Add the DDC ID to blacklist after all checks are passed and trigger the event;

4.2.3.15 Unfreezing

An Operator can call this API to unfreeze DDCs.

- Input parameter: DDC ID
- Output parameter:
- Function name: unFreeze;
- Function example: unFreeze(uint256 ddcId);
- Event: ExitBlacklist(sender, ddcId);
- Core logic:
  - Check whether the caller account state is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the caller account is Operator. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDCID is frozen. If it is, return a prompt message;
  - Remove the DDC ID to blacklist after all checks are passed and trigger the event;

4.2.3.16 Destruction

A DDC owner or authorized account can call this API to destroy DDCs.

- Input parameter: DDC ID;
- Output parameter:
- Function name: burn;
- Function example: burn(uint256 ddcId);
- Event: Transfer(operator, owner, address(0), ddcId);
- Core logic:

- Check whether the caller account is available. If not, return a prompt message;
- Check whether the caller account has the permission. If not, return a prompt message;
- Check whether the DDC ID exists. If not, return a prompt message;
- Check whether the DDC owner account and caller account or the authorized account and caller account belong to the same account. Or check whether the DDC caller account is authorized by the owner account. If they are not the same or the caller account does not have the permission, return a prompt message;
- Call the billing contract to pay for DDC service fee, save DDC destruction data, and remove the DDC ID authorization list and trigger the event;

4.2.3.17 Batch Destruction

A DDC owner or authorized account can call this API to destroy DDCs in a batch.

- Input parameter: DDC ID;
- Output parameter:
- Function name: burnBatch;
- Function example: burnBatch(address owner,uint256[] memory ddcIds);
- Event: TransferBatch(operator, owner, address(0),ddcIds);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;

- Check whether the owner account address is “0 address”. If it is, return a prompt message;
- Check whether the owner account is available. If not, return a prompt message;
- Check whether the DDC owner account and caller account belong to the same platform or whether the DDC owner has authorized all DDCs to the caller. If not, return a prompt message;
- Batch loop to process DDC destruction one by one, the processing logic is as follows:

  1. Check whether the DDC ID exists according to the index. If not, return a prompt message.

  2. Check whether the owner of the DDC is the caller account, or whether the caller account is authorized or whether the DDC owner has authorized all DDCs to the caller according to the index. If not, return a prompt message.

  3. After all checks are passed, call the billing contract to pay the DDC service fee and save the DDC destruction data and remove the DDC ID authorization list.
- Trigger the event.

4.2.3.18 Query Quantity

An Operator, Platform Owner, or end-user can call this API to query the quantity of DDCs owned by the current account.

- Input parameter: owner account;
- Output parameter: Number of DDCs;
- Function name: balanceOf;
- Function example: balanceOf(address owner) view returns (uint256);
- Core logic:

- Check whether the owner address is “0 address”. If it is, return a prompt message;
- The result will be returned after all checks are passed.

4.2.3.19 Batch Query Quantity

An Operator, Platform Owner, or end-user can call this API to query the quantity of DDCs owned by the current account.

- Input parameter: owner account set;
- Output parameter: Set of Number of DDCs;
- Function name: balanceOfBatch;
- Function example: balanceOfBatch(address[] memory owners) view returns (uint256[] memory);
- Core logic:
  - Batch loop to query the number of DDCs one by one, check whether the address of the owner account address is "0 address" according to the index. If it is, return a prompt message. Otherwise, it will query the number of DDCs owned by the corresponding owner account one by one and add them to the list of the number of DDCs owned by the owner account, and return the result.

4.2.3.20 Query the Owner

An Operator, Platform Owner, or end-user can call this API to query the owner of current DDC.

- Input parameter: DDC ID;
- Outputs parameter: owner account;
- Function name: ownerOf;
- Function example: ownerOf(uint256 ddcId) returns (address);
- Core logic:

- Check whether the owner address is “0 address”. If it is, return a prompt message;
- The result will be returned after all checks are passed.

4.2.3.21 Batch Query the Owner

An Operator, Platform Owner, or end-user can call this API to query the owner of current DDC.

- Input parameter: DDC ID;
- Outputs parameter: owner account;
- Function name: ownerOfBatch;
- Function example: ownerOfBatch(uint256[] memory ddcIds) view returns (address[] memory);
- Core logic:
  - Check whether the owner address is “0 address”. If it is, return a prompt message;
  - The result will be returned after all checks are passed.

4.2.3.22 Query DDC Name

An Operator, Platform Owner, or end-user can call this API to query the name of current DDC.

- Input parameter:
- Output parameter: DDC name;
- Function name: name;
- Function example: name() view returns (string memory);
- Core logic:
  - Return a global name.

4.2.3.23 Query Symbol

An Operator, Platform Owner, or end-user can call this API to query the symbol of current DDC.

- Input parameter:
- Output parameter: DDC symbol;
- Function name: symbol;
- Function example: symbol() view returns (string memory);
- Core logic:
  - Return a global symbol.

4.2.3.24 Query ddcURI

An Operator, Platform Owner, or end user can call this function to query the ddcURI of the current DDC.

- Input parameter: DDC ID;
- Output parameter: URI;
- Function name: ddcURI;
- Function example: ddcURI(uint256 ddcId) view returns (string memory);
- Core logic:
  - Check whether the DDC ID exists. If not, return a prompt message;
  - The query result will be returned after all checks are passed.

4.2.3.25 Set ddcURI

An Operator, Platform Owner, or end-user can call this function to set the ddcURI of current DDC.

- Input parameter: DDC ID, URI;
- Output parameter:
- Function name: setURI;

- Function example: setURI(uint256 ddcId, string memory ddcURI);
- Event: SetURI(address indexed operator,uint256 indexed ddcId, string ddcURI);
- Core logic:
  - Check whether the URI corresponding to the DDC ID is null. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the URI that is going to be uploaded is null. If it is, return a prompt message;
  - Check whether the caller's address exists. If not, return a prompt message;
  - Check whether the status of the Platform Owner corresponding to the caller address is active. If not, return a prompt message;
  - Check whether the status of the Operator corresponding to the caller address is active. If not, return a prompt message;
  - Check whether the DDC owner account and caller account or the authorized account and caller account belong to the same account. Or check whether the DDC caller account is authorized by the owner account. If they are not the same or the caller account does not have the permission, return a prompt message;
  - The data of the ddcURI will be saved after all checks are passed, and then trigger the event.

4.2.3.26 Set Name and Symbol

Contract owners can initialize the name and symbol of 721-DDCs.

- Input parameter: name, symbol;
- Output parameter:
- Function name: setNameAndSymbol;

- Function example: setNameAndSymbol(string memory name_, string memory symbol_);
- Event: SetNameAndSymbol(string name, string symbol);
- Core logic:
  - Check whether the caller account is the contract owner. If not, return a prompt message;
  - Save name and symbol after all checks are passed, and trigger the event.

4.2.3.27 Query the Latest DDC ID

An Operator can call this function to get the latest DDC ID.

- Input parameter:
- Output parameter: latest DDC ID;
- Function name: getLatestDDCId;
- Function example: getLatestDDCId() view returns (uint256);
- Core logic:
  - Return the latest DDC ID.

4.3 BSN-DDC-1155 Business Contract

4.3.1 Function Description

The BSN-DDC-1155 main contract is used to provide chain accounts with a set of APIs, including BSN-DDC generation, batch generation, authorization, query authorization, transfer, batch transfer, freezing, unfreezing, and destruction.

4.3.2 Data Structure

- DDC storage structure

No.	Field name	Field	Type	Remarks

1.	DDC URI set	_ddcURIs	mapping<uint256=>string>	Key: DDC ID Value: URI
2.	DDC balance mapping set	_balances	mapping(address => uint256)	Key: DDC ID Value: Owner address and number of DDCs
3.	DDC operator mapping set	_operatorApprovals	mapping(address => mapping(address => bool))	Key: DDC owner address Value: authorized account address and option whether to authorize
4.	DDC blacklist	_blacklist	mapping<uint256=>bool>	Key: DDC ID Value: DDC state
5.	DDC ID List	_ddcIds	mapping<uint256=>bool>	Key: DDC ID Value: DDC ID exists or not
6.	The latest DDC ID	_lastDDCId	uint256	

4.3.3 API Definition

4.3.3.1 Safe Generation

A Platform Owner or an end-user can call this API to generate a DDC.

- Input parameters: receiver account, amount of copies of the DDC, URI, additional data;
- Output parameter:
- Function name: safeMint;
- Function example: safeMint(address to,uint256 amount, string memory ddcURI,bytes memory data ;
- Event: TransferSingle(operator, address(0),to, ddcId, amount);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;

- Check whether the receiver account is available. If not, return a prompt message;
- Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
- Check whether the generated DDC amount is greater than 0. If not, return a prompt message;
- Generate the DDC ID and check whether the DDC ID already exists. If it does, return a prompt message;
- Call the billing contract to pay the DDC service fee after all checks are passed and save the generated DDC data;
- Trigger the event, check whether the receiver account is a contract. If it is, check whether this receiver account can receive DDCs.

4.3.3.2 Batch Generation

A Platform Owner or an end-user can call this API to generate DDCs in a batch.

- Input parameters: receiver account, amount set, URI set, additional data;
- Output parameter:
- Function name: safeMintBatch;
- Function example: safeMintBatch(address to,uint256[] memory amounts, string[] memory ddcURIs, bytes memory data);
- Event: TransferBatch(operator, address(0), to, ddcIds, amounts);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has call permission. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;

- Check whether the receiver account is available. If not, return a prompt message;
- Check whether the caller account and receiver account belong to the same platform. If not, return a prompt message;
- Check whether the length of data set is equal to that of URI set. If not, return a prompt message;
- Batch loop to process DDCs and check whether the DDC amount is greater than 0 after query. If not, return a prompt message. Check whether the generated DDC ID exists. If not, return a prompt message. Or call the billing contract separately to pay for DDC service fee and save the generated DDC data;
- Trigger the event, check whether the receiver account is a contract. If it is, check whether this receiver account can receive DDCs.

4.3.3.3 Account Authorization

A DDC owner can call this API to authorize an account.

- Input parameters: authorized account, authorization identifier
- Output parameter:
- Function name: setApprovalForAll;
- Function example: setApprovalForAll(address operator, bool approved);
- Event: ApprovalForAll(owner, operator, approved);
- Core logic:
  - Check whether the caller is available. If not, return a prompt message;
  - Check whether the caller has call permission. If not, return a prompt message;
  - Check whether the authorized account address is “0 address”. If it is, return a prompt message;
  - Check whether the authorized account is available. If not, return a prompt message;

- Check whether the caller account and the authorized account belong to the same platform. If not, return a prompt message;
- Check whether the DDC owner account and the authorized account are the same account. If they are, return a prompt message;
- Call the billing contract to pay for DDC service fee after all checks are passed and save the DDC authorization data and trigger the event;
- Note: Only DDCs of the current owner can be authorized. The authorization will be invalid after transfer;

4.3.3.4 Verify Account Authorization

An Operator, Platform Owner, or end-user can call this API to verify the account authorization.

- Input parameters: owner account, authorized account
- Output parameter: bool result;
- Function name: isApprovedForAll;
- Function example: isApprovedForAll(address owner, address operator) view returns (bool);
- Core logic:
  - Check whether the owner account address or the authorized account address is “0 address”. If it is, return a prompt message;
  - Check whether the authorized account is authorized by the owner account and return the validation result;

4.3.3.5 Safe Transfer

A DDC owner or an authorized account can call this API to securely transfer DDCs.

- Input parameters: owner address, receiver account, DDC ID, amount of DDCs, additional data;

- Output parameter:
- Function name: safeTransferFrom;
- Function example: safeTransferFrom(address from, address to, uint256 ddcId, uint256 amount, bytes memory data);
- Event: TransferSingle(operator, from, to, ddcId, amount);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has permission. If not, return a prompt message;
  - Check whether the owner account is available. If not, return a prompt message;
  - Check whether the owner account address is “0 address”. If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;
  - Check whether the receiver account address is “0 address”. If it is, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDCID is frozen. If it is, return a prompt message;
  - Check whether the DDC owner account and the receiver account belong to the same platform or is authorized by another platform. If not, return a prompt message;
  - Check whether the DDC owner account and the caller account are the same account. Or check whether the DDC caller account is authorized by the owner account. If not, return a prompt message;
  - Check whether the amount of safe transferred DDCs is greater than 0. If not, return a prompt message;

- Check whether the owner's DDC amount is equal to or greater than that of DDCs to be transferred. If not, return a prompt message;
- Call the billing contract to pay the DDC service fee after all checks are passed, and save the DDC safe transfer data;
- Trigger the event, check whether the receiver account is a contract. If it is, check whether this receiver account can receive DDCs.

4.3.3.6 Batch Safe Transfer

A DDC owner or an authorized account can call this API to transfer DDCs in a batch.

- Input parameters: owner account, receiver account, DDC ID set, number set, additional data
- Output parameter:
- Function name: safeBatchTransferFrom;
- Function example: safeBatchTransferFrom(address from, address to,uint256[] memory ddcIds,uint256[] memory amounts, bytes memory data);
- Event: TransferBatch(operator, from, to, ddcIds, amounts);
- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the owner account is available. If not, return a prompt message;
  - Check whether the owner account address is "0 address". If it is, return a prompt message;
  - Check whether the receiver account is available. If not, return a prompt message;

- Check whether the receiver account address is “0 address”. If it is, return a prompt message;
- Check whether the DDC owner account and the receiver account belong to the same platform or is authorized by another platform. If not, return a prompt message;
- Check whether the length of the data set is equal to that of the URI set. If not, return a prompt message;
- Batch loop to process DDCs and check whether each DDC ID exists. If not, return a prompt message. Check whether each DDC is unfrozen. If not, return a prompt message. Check whether the amount of DDCs to be transferred is greater than 0. If not, return a prompt message. Check the amount of DDC in the owner account is equal to or greater than the amount of DDCs to be transferred. If not, return a prompt message. If it is, then separately call the billing contract to pay for DDC service fee and save the DDC safe transfer data.
- Trigger the event, check whether the receiver account is a contract. If it is, check whether this receiver account can receive DDCs.

4.3.3.7 Freezing

An Operator can call this API to freeze DDCs.

- Input parameter: DDC ID;
- Output parameter:
- Function name: freeze;
- Function example: freeze(uint256 ddcId);
- Event: EnterBlacklist(sender, ddcId);
- Core logic:
  - Check whether the caller account state is available. If not, return a prompt message;

- Check whether the caller account has the permission. If not, return a prompt message;
- Check whether the caller account is Operator. If not, return a prompt message;
- Check whether the DDC ID exists. If not, return a prompt message;
- Check whether the DDCID is frozen. If it is, return a prompt message;
- Add the DDC ID to the blacklist after all checks are passed and trigger the event;

4.3.3.8 Unfreezing

An Operator can call this API to unfreeze DDCs.

- Input parameter: DDC ID
- Output parameter:
- Function name: unFreeze;
- Function example: unFreeze(uint256 ddcId);
- Event: ExitBlacklist(sender, ddcId);
- Core logic:
  - Check whether the caller account state is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the caller account is Operator. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDCID is frozen. If it is, return a prompt message;
  - Remove the DDC ID from the blacklist after all checks are passed and trigger the event;

4.3.3.9 Destruction

A DDC owner can call this API to destroy DDCs.

- Input parameter: DDC owner account, DDC ID;
- Output parameter:
- Function name: burn;
- Function example: burn(address owner, uint256 ddcId);
- Event: TransferSingle(operator, owner, address(0), ddcId, amount);
- Core logic:
- Check whether the caller account is available. If not, return a prompt message;
- Check whether the caller account has the permission. If not, return a prompt message;
- Check whether the DDC ID exists. If not, return a prompt message;
- Check whether the owner's DDC amount is greater than 0. If not, return a prompt message;
- Check whether the DDC owner account and the caller account are the same account. Or check whether the DDC caller account is authorized by the owner account. If not, return a prompt message;
- Call the billing contract to pay the DDC service fee after all checks are passed and save the destroyed DDC data and trigger the event;

4.3.3.10 Batch Destruction

The DDC owner can call this API to destroy DDCs in a batch.

- Input parameters: owner account, DDC ID set
- Output parameter:
- Function name: burnBatch;
- Function example: burnBatch(address owner,uint256[] memory ddcIds);
- Event: TransferBatch(operator, owner, address(0), ddcIds, amounts);

- Core logic:
  - Check whether the caller account is available. If not, return a prompt message;
  - Check whether the caller account has the permission. If not, return a prompt message;
  - Check whether the DDC owner account and the caller account are the same account. Or check whether the owner has authorized all the DDC to the caller account. If not, return a prompt message;
  - Loop through the DDC list in batch and check whether the DDC ID exists. If not, return a prompt message. Check whether the amount of the owner's DDCs is greater than 0. If not, return a prompt message. If it is, then call the billing contract to pay for DDC service fee and save the destroyed DDC data;
  - Trigger the event;

4.3.3.11 Query Number

An Operator, Platform Owner, or end-user can call this API to query the number of DDCs owned by the current account.

- Input parameters: owner account, DDC ID
- Output parameter: balance
- Function name: balanceOf;
- Function example: balanceOf(address owner,uint256 ddcId) view returns (uint256);
- Core logic:
  - Check whether the owner address is "0 address". If it is, return a prompt message;
  - The result will be returned after all checks are passed;

4.3.3.12 Batch Query Number

An Operator, Platform Owner, or end-user can call this API to query the number of DDCs owned by the current account.

- Input parameters: owner account set, DDC ID set
- Output parameter: balance set
- Function name: balanceOfBatch;
- Function example: balanceOfBatch(address[] memory owners,uint256[] memory ddcIds) view returns (uint256[] memory);
- Core logic:
  - Check whether the length of data set is equal to that of URI set. If not, return a prompt message;
  - Loop through the DDC list in batch and check whether the owner address is “0 address”. If it is, return a prompt message. Otherwise, the number of DDCs corresponding to each query will be added one by one to the set list and return the results.

4.3.3.13 Query ddcURI

An Operator, Platform Owner, or end user can call this function to query the ddcURI of the current DDC.

- Input parameter: DDC ID;
- Output parameter: URI;
- Function name: ddcURI;
- Function example: ddcURI(uint256 ddcId) view returns (string memory);
- Core logic:
  - Check whether the DDC ID exists. If not, return a prompt message;
  - The query result will be returned after all checks are passed;

4.3.1.14 Set ddcURI

A DDC owner or an authorized account can call this function to set the ddcURI of current DDC.

- Input parameter: DDC ID, URI;
- Output parameter:
- Function name: setURI;
- Function example: setURI(address owner,uint256 ddcId,string memory ddcURI);
- Event: SetURI(address indexed operator, address indexed owner,uint256 indexed ddcId, string ddcURI);
- Core logic:
  - Check whether the caller account exists. If not, return a prompt message;
  - Check whether the status of the Platform Owner corresponding to the caller address is active. If not, return a prompt message;
  - Check whether the status of the Operator corresponding to the caller address is active. If not, return a prompt message;
  - Check whether the caller account has permission. If not, return a prompt message;
  - Check whether the DDC ID exists. If not, return a prompt message;
  - Check whether the DDC ID available. If not, return a prompt message;
  - Check whether the owner account and the caller account are the same, or the owner account has authorized the caller account. If not, return a prompt message;
  - Check whether the DDC URI is null. If it is, return a prompt message;
  - Check whether the URI corresponding to the DDC ID is null. If it is, return a prompt message;
  - Save the ddcURI after all checks are passed, and then trigger the event;

4.3.1.15 Query the Latest DDC ID

An Operator can call this function to query the latest DDC ID.

- Input parameter:
- Output parameter: latest DDC ID;
- Function name: getLatestDDCId;
- Function example: getLatestDDCId() view returns (uint256);
- Core logic:
  - Return the latest DDC ID;

4.4 BSN-DDC Authority Contract

4.4.1 Function Description

The BSN-DDC-Authority contract is used to implement the logic of the role for a certain account. There are three kinds of authority roles: Operator, Platform, and End-user. Different roles have different authority, and a real entity can hold multiple chain accounts to have multiple roles in the BSN-DDC Network, such as the BSN can be the Operator in the BSN-DDC Network. The three roles are the leaderAddress of the subordinate role, which can manage the subordinate account.

- End-user authority: End-users can manage the entire life cycle of DDCs after being bound to a platform.
- Platform Owner authority: Platform Owners can manage the entire life cycle of DDCs and the addition, deletion, modification, and query of user accounts on the platform.
- Operator authority: Operators have the permission to add, delete, modify, and query the accounts of End-users, Operators, as well as Platform Owners. They can fix a price for the DDC transfers, including the fees of DDC generation and transfer. In addition, Operators can top up the accounts of Platform Owners.

4.4.2 Data Structure

- BSN-DDC-authority management-account information

No.	Field name	Field	Type	Remarks
1.	Private key address	account	address	DDC user chain account address
2.	Account information	_accountInfo	AccountInfo	DDC account information
AccountInfo				
1.	Account DID	accountDID	string	DID information of DDC account (it can be null for common user)
2.	Account name	accountName	string	DDC account information
3.	Account role	accountRole	enum	Identity information of DDC account. The value includes: 0. Operator (Operator) 1. PlatformManager (Platform Owner) 2. Consumer (End-user)
4.	Account leader manager	leaderDID	string	Leader manager of the DDC account; it is mandatory when the account role is Consumer. For general Consumer, this value is PlatformManager
5.	State of Platform management account	platformState	enum	The current DDC account state (this state can be operated by Platform Owner only). The value includes: 1. Frozen (frozen state, DDC related operations are unavailable) 2. Active (active state, DDC related operations are available)
6.	State of Operator management account state	operatorState	enum	The current DDC account state (this state can be operated by the operator only). The value includes:

				1. Frozen (frozen state, DDC related operations unavailable) 2. Active (active state, DDC related operations available)
7.	Redundancy field	field	string	Redundancy field

- BSN-DDC-authority management-function information

No.	Field name	Field	Type	Remarks
1.	Account role	role	string	Identity information of DDC account. The value includes: 1. Operator (Operator) 2. PlatformManager (Platform Owner) 3. Consumer (End-user)
2.	Contract function	_funAclList	FunAcl[]	Contract function
FunAcl				
1.	Contract address	contractAddress	address	DDC contract address
2.	Function list	funList	String[]	The accessible function list of DDC roles
3.	Index list	sigIndexList	mapping(bytes4 => uint256)	The query list of contract function signature

- BSN-DDC-authority management-contract index list

No.	Field name	Field	Type	Remarks
1.	Index list	_contractIndexList	mapping(Role => mapping(address => uint256))	The contract index list of account type

- BSN-DDC-authority management-account DID authorization list

No.	Field name	Field	Type	Remarks
1.	Account DID authorization list	_didApprovals	mapping(string => mapping(string => bool))	Account DID authorization set Key: Authorizer account DID Value: receiver account DID->authorized or not

- BSN-DDC-authority management- Platform Owner DID set

No.	Field name	Field	Type	Remarks

1.	Platform Owner DID set	_platformDIDs	mapping(string => bool)	Platform Owner DID set Key：Platformer DID Value: whether it exists or not

- BSN-DDC-authority management- Platform Owner adds chain account switch

No.	Field name	Field	Type	Remarks
1.	Whether to open	_platformSwitcher	bool	

4.4.3 API Definition

4.4.3.1 Add Operator Account

A contract owner can call this API to add an Operator account.

- Input parameters: account address, account name, account DID;
- Output parameter:
- Function name: addOperator;
- Function example: addOperator (address operator, string memory accountName, string memory accountDID);
- Event: AddAccount (address indexed caller, address indexed account);
- Core logic:
  - Check whether the caller is the contract owner. If not, return a prompt message;
  - Check whether the DID length is “0 address”. If it is, return a prompt message;
  - Check whether the account name length is “0 address”. If it is, return a prompt message;
  - Check whether the Operator account to be added already exists. If it does, return a prompt message;
  - Save Operator account data after all checks are passed and trigger the event.

4.4.3.2 Add Account by Platform Owners

Platform Owners can create DDC account information by calling this API. “address” is the user's chain account address. “AccountInfo” refers to the user's detailed information. The leader account can operate its subordinate accounts. Platform Owners can only add end-user accounts through this function.

- Input parameter: account address, account name, account DID;
- Output parameter:
- Function name: addAccountByPlatform;
- Function example: addAccountByPlatform(address account, string memory accountName, string memory accountDID);
- Event: AddAccount (address indexed caller, address indexed account);
- Core logic:
  - Check whether the caller exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is Platform Owner. If not, return a prompt message;
  - Check whether the length of the name of the account to be added is 0. If it is, return a prompt message;
  - Check whether the account to be added already exists. If not, return a prompt message;
  - Save account data after all checks are passed and trigger the event.

4.4.3.3 Batch Add Account by Platform Owners

Platform Owners can create DDC account information by calling this API. “address” is the user's chain account address. “AccountInfo” refers to the user's

detailed information. The leader account can operate its subordinate accounts. Platform Owners can only add end-user accounts in a batch through this function.

- Input parameter: account address set, account name set, account DID set;
- Output parameter:
- Function name: addBatchAccountByPlatform;
- Function example: addBatchAccountByPlatform(address[] memory accounts,string[] memory accountNames, string[] memory accountDIDs);
- Event: AddBatchAccount(address indexed caller, address[] indexed accounts);
- Core logic:
  - Check whether the caller exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is Platform Owner. If not, return a prompt message;
  - Check whether the length of the account address set, the length of the account name set and the length of the account DID set are equal. If not, return a prompt message;
  - Batch loop to process the addition of chain account, the processing logic is as follows:

    1. Check whether the caller account is open to add chain account. If not, return a prompt message;

    2. Check whether the length of the added account name is 0 according to the index. If it is, return a prompt message;

    3. Check whether the account to be added already exists according to the index. If it does, return a prompt message;

    4. Save the account data after all checks are passed.
  - Trigger the event.

4.4.3.4 Set the Switch to added Chain Account by Platform Owners

An Operator can call this API to set the switch authority control of adding chain account.

- Input parameter: whether to open;
- Output parameter:
- Function name: setSwitcherStateOfPlatform;
- Function example: setSwitcherStateOfPlatform(bool isOpen);
- Event: SetSwitcherStateOfPlatform(address indexed operator, bool isOpen);
- Core logic:
  - Check whether the caller exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is Operator. If not, return a prompt message;
  - Check whether isOpen is the same with the switch status. If it is, return a prompt message;
  - Save account data after all checks are passed and trigger the event.

4.4.3.5 Query the Switch of the added Chain Account by Platform Owners

An Operator, Platform Owner and end-user can call this API to set the switch authority control of adding chain account.

- Input parameter:
- Output parameter: bool result
- Function name: switcherStateOfPlatform;
- Function example: switcherStateOfPlatform() view returns (bool);

- Core logic:
  - Return the status of the switch of adding chain account by the Platform Owner;

4.4.3.6 Add Account by Operator

An Operator can create accounts for Platform Owners or their end-users by calling this API.

- Input parameters: account address, account name, account DID, leader DID;
- Output parameter:
- Function name: addAccountByOperator;
- Function example: addAccountByOperator(address account, string memory accountName, string memory accountDID, leaderDID);
- Event: AddAccount (address indexed caller, address indexed account);
- Core logic:
  - Check whether the caller exist. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is Operator. If not, return a prompt message;
  - Check whether the length of the added account name is 0. If it is, return a prompt message;
  - Check whether the account to be added already exists. If it does, return a prompt message;
  - Check whether the leader DID is empty. If not, check whether the Platform Owner corresponding to the leader DID exists. If not, return a

prompt message. If the leader DID is empty, check whether account DID is empty. If it is, return a prompt message.

- Save account data after all checks are passed and trigger the event.
- Note: If the leader DID of the added account is empty, it means that the added account is the Platform Owner.

4.4.3.7 Batch Add Accounts by Operator

An Operator can create accounts in a batch for Platform Owners or their end-users by calling this API.

- Input parameters: account address set, account name set, account DID set, leader DID set;
- Output parameter:
- Function name: addBatchAccountByOperator;
- Function example: addBatchAccountByOperator(address[] memory accounts, string[] memory accountNames,string[] memory accountDIDs, string[] memory leaderDIDs);
- Event: AddBatchAccount(address indexed caller, address[] indexed accounts);
- Core logic:
  - Check whether the caller exist. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is the Operator. If not, return a prompt message;
  - Check whether the length of the account address set, the length of the account name set, the length of the account DID set and the length of the leader DID set are equal. If not, return a prompt message;

- Batch loop to process the addition of chain account, the processing logic is as follows:

  1. Check whether the length of the added account name is 0 according to the index. If it is, return a prompt message;

  2. Check whether the account to be added already exists according to the index. If it does, return a prompt message;

  3. Check whether the leader DID is empty. If not, check whether the Platform Owner corresponding to the leader DID exists. If not, return a prompt message. If the leader DID is empty, check whether the account DID is empty. If it is, return a prompt message;

  4. Save the account data after all checks are passed.
- Trigger the event. Note: If the leader DID of the added account is empty, it means that the added account is the Platform Owner.

4.4.3.8 Update Account Status

An Operator or Platform Owner can change the DDC account information status via this API. The address is the user chain account address. A Platform Owner can freeze or unfreeze the Platform status identifier of end-users while an Operator can freeze or unfreeze the Platform status identifier and the Operator status identifier of end-users and platforms.

- Input parameters: account address, account status, update Platform Owner status or not;
- Output parameter:
- Function name: updateAccountState;
- Function example: updateAccountState(address account, State state, bool changePlatformState);
- Event: UpdateAccountState(address indexed account, IAuthorityData.State platformState, IAuthorityData.State operatorState);
- Core logic:

- Check whether the account exists. If not, return a prompt message;
- Check whether the caller account exists. If not, return a prompt message;
- Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
- Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
- Check whether the account leader DID is the same as the caller account DID. Or check whether the caller account is Operator. If not, return a prompt message;
- If the caller account is the Operator:

  1. If "changePlatformState" is true, check whether "state" is the same with Platform Owner status. If it is, return a prompt message;

  2. If "changePlatformState" is false, check whether "state" is the same with Operator status. If it is, return a prompt message;
- If the caller account is the Platform Owner, check whether the Platform Owner state of the account is the same with "state". If it is, return a prompt message;
- Update the account status after all checks are passed and trigger the event.

4.4.3.9 Query Account

An Operator or Platform Owner can change the DDC account information status via this API. The address is the user chain account address. The leader role can manage the subordinate accounts.

- Input parameter: account address;
- Output parameter: account information;
- Function name: getAccount;
- Function example: getAccount(address account) view returns (AccountInfo);

- Core logic:
  - Return query results after all checks are passed.

4.4.3.10 Delete Account

An Operator can delete DDC account information via this API. The address is the user chain account address. The leader role can manage the subordinate accounts. Currently, this function cannot be called externally.

- Input parameter: account address;
- Output parameter:
- Function name: delAccount;
- Function example: delAccount(address account);
- Event: DelAccount(address indexed account);
- Core logic:
  - Check whether the account to be deleted exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the leader role of the account to be deleted is the caller by matching the account's leader DID and the caller account DID. If it does not match, return a prompt message;
  - Delete account data after all checks are passed and trigger the event;

4.4.3.11 Verify Account Status

A BSN-DDC-721/1155 proxy contract can check whether the account corresponding to a chain account address is available via this API.

- Input parameter: account address;

- Output parameter: account role;
- Function name: accountAvaliable;
- Function example: accountAvaliable(address account) view returns (bool);
- Core logic:
  - Call the data contract to check whether the account exists. If not, return a prompt message;
  - Check whether the Platform Owner status of the account is active. If not, return a prompt message;
  - Check whether the Operator status of the account is active. If not, return a prompt message;
  - Return the availability of the account after all checks are passed;

4.4.3.12 Role Assertion

A BSN-DDC-721/1155 proxy contract can assert the role of an account via this API.

- Input parameters: account address, account role;
- Output parameter: bool result;
- Function name: checkAvailableAndRole ;
- Function example: checkAvailableAndRole (address account, Role role) view returns (bool);
- Core logic:
  - Call the data contract to check whether the account exists. If not, return a prompt message;
  - Check whether the Platform Owner status of the account is active. If not, return a prompt message;
  - Check whether the Operator status of the account is active. If not, return a prompt message;
  - Return the account type and role comparison results after all checks are passed;

4.4.3.13 Same Platform Verification

A BSN-DDC-721/1155 proxy contract can verify whether two accounts belong to the same platform by calling this API.

- Input parameters: account 1, account 2;
- Output parameter: bool result;
- Function name: onePlatformCheck;
- Function example: onePlatformCheck(address acc1, address acc2) view returns (bool);
- Core logic:
  - Check whether account 1 exists. If not, return a prompt message;
  - Check whether the Platform Owner to which account 1 belongs is active. If not, return a prompt message;
  - Check whether the Operator to which account 1 belongs is active. If not, return a prompt message;
  - Check whether account 2 exists. If not, return a prompt message;
  - Check whether the Platform Owner to which account 2 belongs is active. If not, return a prompt message;
  - Check whether the Operator to which account 2 belongs is active. If not, return a prompt message;
  - If the roles of account 1 and account 2 are both Platform Owners, then check whether the account DIDs of both accounts match and then check whether the leader DIDs of both accounts match. If they do, return the result “true”. If they do not match, return a prompt message;
  - If the role of account 1 is Platform Owner while that of account 2 is end-user, then check whether the account DID of account 1 and the leader DID of account 2 match. If they do, return the result “true”. If they do not match, return a prompt message ;
  - If the role of account 1 is end-user while that of account 2 is Platform Owner, then check whether the leader DID of account 1 and the account

DID of account 2 match. If they do, return the result “true”. If they do not match, return a prompt message ;

- If the roles of account 1 and account 2 are both end-users, then check whether the leader DIDs of both accounts match. If they do, return the result “true”. If they do not match, return a prompt message ;
- Return “false” if there are other results;

4.4.3.14 Cross-Platform Verification

A BSN-DDC-721/1155 proxy contract can verify whether two accounts are cross-platform accounts by calling this API.

- Input parameters: authorizer account, receiver account;
- Output parameter: bool result;
- Function name: crossPlatformCheck;
- Function example: crossPlatformCheck(address from, address to) view returns (bool);
- Core logic:
  - Check whether authorizer account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the authorizer account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the authorizer account belongs is active. If not, return a prompt message;
  - Check whether the receiver account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the receiver account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the receiver account belongs is active. If not, return a prompt message;

- If the roles of the authorizer account and the receiver account are both Platform Owners, then check whether the account DIDs of both accounts match and then check whether the leader DIDs of both accounts are different. If they do not meet these requirements, return a prompt message. If they do, check whether the account DIDs of both accounts are in the authorization list. If they are, return the result “true”. If not, return a prompt message;
- If the role of the authorizer account is Platform Owner while that of the receiver account is end-user, then check whether the account DID of the authorizer account and the leader DID of the receiver account are in the authorization list. If they are, return the result “true”. If not, return a prompt message;
- If the role of the authorizer account is end-user while that of the receiver account is Platform Owner, then check whether the leader DID of the authorizer account and the account DID of the receiver account are in the authorization list. If they are, return the result “true”. If not, return a prompt message;
- If the roles of the authorizer account and the receiver account are both end-users, then check whether the leader DIDs of both accounts are in the authorization list. If they do, return the result “true”. If they do not match, return a prompt message;
- Return “false” if there are other results;

4.4.3.15 Add Function

An Operator can add the functions that can be called by roles via this API.

- Input parameters: account role, contract address, function name;
- Output parameter: bool result;
- Function name: addFunction;

- Function example: addFunction(Role role, address contractAddress, byte4 sig) returns (bool);
- Event: AddFunction(address indexed operator, Role indexed role, address contractAddress,byte4 sig);
- Core logic:
  - Check whether the caller exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller is Operator. If not, return a prompt message;
  - Save all function data of the proxy contract after all checks are passed and trigger the event.

4.4.3.16 Delete Function

An Operator can delete the functions that can be called by roles via this API.

- Input parameters: account role, contract address, function name;
- Output parameter:
- Function name: delFunction;
- Function example: delFunction(Role role, address contractAddress, bytes4 sig) returns (bool);
- Event: DelFunction(address indexed operator, Role indexed role, address contractAddress,byte4 sig);
- Core logic:
  - Check whether the caller exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;

- Check whether the caller is Operator. If not, return a prompt message;
- Check whether the function to be deleted exists. If not, return a prompt message;
- Delete all function data of the proxy contract after all checks are passed and trigger the event.

4.4.3.17 Query Function

An Operator can query this function to call different roles via this API.

- Input parameters: account role, contract address;
- Output parameter: function list;
- Function name: getFunctions;
- Function example: getFunctions(Role role, address contractAddress) view returns (bytes4[] memory);
- Core logic:
  - Return query results;

4.4.3.18 Verify Function Authority

A BSN-DDC-721/1155 proxy contract can verify the authority among the caller account, the called contract and the called function.

- Input parameters: account address, contract address, function name;
- Output parameter: bool result;
- Function name: hasFunctionPermission;
- Function example: hasFunctionPermission(address account, address contractAddress, bytes4 sig) view returns (bool);
- Core logic:
  - Check whether the account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the account belongs is active. If not, return a prompt message;

- Check whether the Operator to which the account belongs is active. If not, return a prompt message;
- According to the account type and the function list to get the contract, check whether the uploaded signature function is on the function list and return the result.

4.4.3.19 Cross-Platform Authorization

An Operator calls this API for cross-platform account authorization.

- Input parameters: authorizer account, receiver account, authorization identifier;
- Output parameter:
- Function name: crossPlatformApproval;
- Function example: crossPlatformApproval(address from, address to, bool approved);
- Event: CrossPlatformApproval(address indexed from, address indexed to, bool approved);
- Core logic:
  - Check whether the caller account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the account belongs is active. If not, return a prompt message;
  - Check whether the caller account is the Operator. If not, return a prompt message;
  - Check whether the authorizer account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the authorizer account belongs is active. If not, return a prompt message;

- Check whether the Operator to which the authorizer account belongs is active. If not, return a prompt message;
- Check whether the receiver account is the Platform Owner. If not, return a prompt message;
- Check whether the authorizer account and the receiver account belong to the same platform. If they do, return a prompt message;
- Save the cross-platform authorization result after all checks are passed and trigger the event.

4.4.3.20 Sync Platform Owner's DID

An Operator stores the old Platform Owner's DID to the Platform Owner DID set by calling this API.

- Input parameters: whether to open;
- Output parameter:
- Function name: syncPlatformDID;
- Function example: syncPlatformDID(string[] memory dids);
- Event: SyncPlatformDID(address indexed operator, string[] dids);
- Core logic:
  - Check whether the caller account exists. If not, return a prompt message;
  - Check whether the Platform Owner to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the Operator to which the caller account belongs is active. If not, return a prompt message;
  - Check whether the caller account is the Operator. If not, return a prompt message;
  - Batch loop to process the addition of the chain accounts, the processing logic is as follow:

    1. Check whether the DID is empty. If it is, return a prompt message.

2. Check whether the DID exists in the DID set. If it does, return a prompt message.

3. Save the Platform Owner DID after all checks are passed.

- Trigger the event.