Datamall Chain

D M C

Mining Manual

V 1.0.0

August 2022

Content

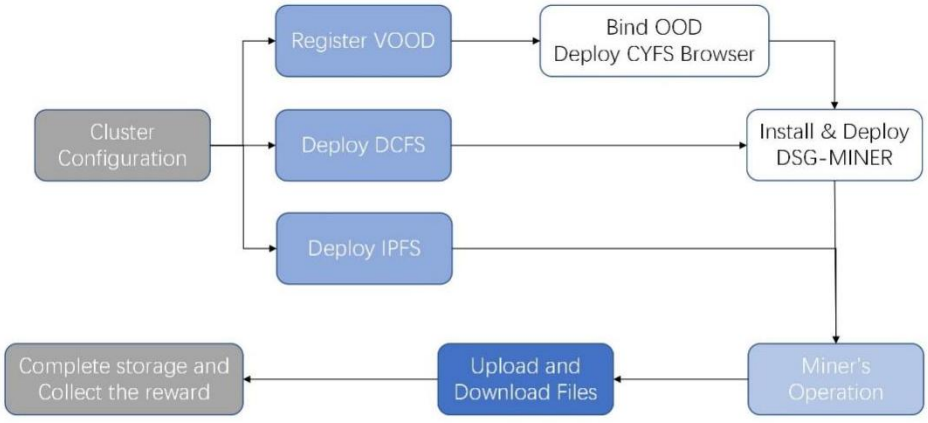
Introduction

Nowadays, with the continuous improvement of mining computing power, in order to maximize the mining revenue, Datamall Chain supports both DCFS mining and IPFS mining. This manual aims to help you better understand the process of DCFS mining and complete the configuration and deployment of DCFS mining.

Based on an open source community, you can always come up with new topics, add new content, and provide examples where you think might be useful.

This manual focus on introducing DCFS, for the configuration and deployment of IPFS, please refer to IPFS Technical Manual.

The following is the procedures of mining on Datamall Chain.



1. Cluster Configuration for Reference

The following is the reference configuration for the cluster of 1 P

Same Configuration for System Machine and Data Machine (2pcs)

CPU: INTEL XEON SILVER 4214 12C/24T 2.2GHz * 2

RAM: DDR4 32G * 4

SSD: SATA SSD 480G *2

SAS: 1.2T * 2

Raid Controller Card : 2G cache

Configuration of Storage Machine (2 pcs)

CPU: INTEL XEON SILVER 4210 * 1

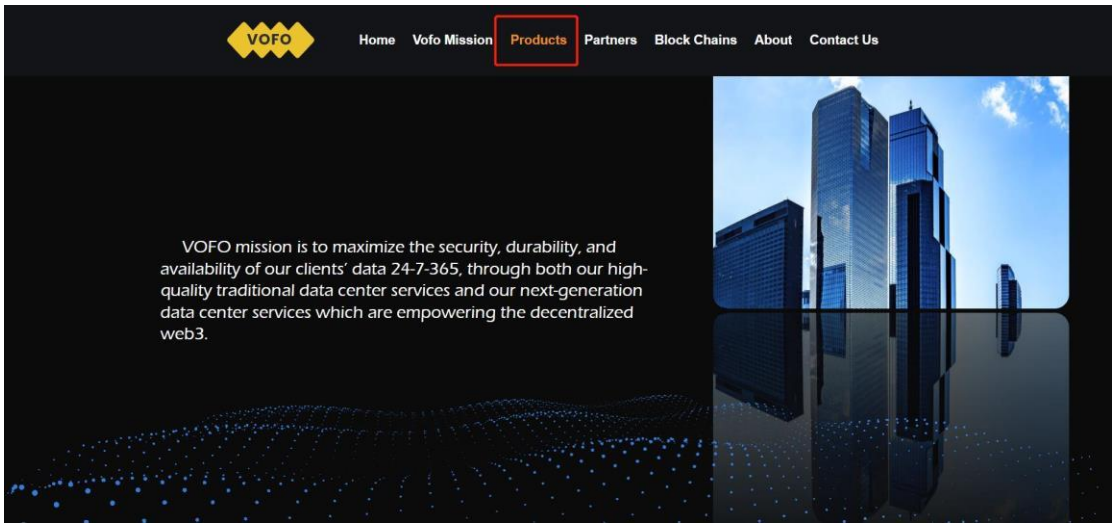
RAM: DDR4 32G * 4

SSD: 480G * 2

HDD: 18T * 36

NET: 25+GB/s bandwidth

2. Register VOOD



VOOD is an innovative product in the DMC ecosystem. V stands for virtual, so VOOD is virtual OOD and no physical device is required. Miners can learn about and rent VOOD virtual machine storage services at http://www.vofocorp.com/#Home.

3. Deploy DCFS

DSFS cluster service list

Build DCFS cluster environment

3.1 Basic Software Environment

Before deploying DCFS cluster, the software service that DCFS depends on should be deployed first.

- [1. Build etcd cluster](./steps of cold deployment / Build etcd cluster .md)
- [2.Build etcd-meta cluster](./ steps of cold deployment / Build etcd cluster .md)
- [3. Build mysql node](./ steps of cold deployment / Build mysql node.md)
- [4. Build and initialize influxDB] (./ steps of cold deployment/Build and initialize influxDB.md)
- [5. Initialize storage node environment](./ steps of cold deployment / Initialize storage node enviroment.md)
- [6. The basic software environment DCFS service node depends on](./ steps of cold deployment /The basic software environment DCFS service node depends on.md)
- [7. Build bundling machine environment] (./ steps of cold deployment / Build bundling machine environment.md)

3.2 Introduction of Cluster Configuration Documents

After building software environment that each component of DCFS depends on, you also need to write cluster configuration documents based on the practical configuration of the cluster.

- External dependencies configuration files： [config.js](./ steps of cold deployment /test_bucky/config.js)
- influxdb configuration files：[fil_perf.toml](./ steps of cold deployment /test_bucky/fil_perf.toml)
- Name of machine cluster host configuration： [machine.js](./ steps of cold deployment /test_bucky/machine.js)
- etcd service： [selector.js](./ steps of cold deployment /test_bucky/selector.js)
- DCFS service node installation configuration： [service_group.js](./ steps of cold deployment /test_bucky/service_group.js)
- DCFS storage machine topology configuration files：[topology.toml](./ steps of cold deployment /test_bucky/topology.toml)

3.3 The Steps of Initial Cold Start of Cluster Deployment

Detail operation for reference: [The steps of Initial cold start of cluster deployment](./ steps of cold deployment/ the steps of bundling machine cold deployment.md)

3.4 Cluster Monitoring Configuration

- DCFS panels configuration: [DCFS monitoring panels configuration.md] (./steps of cold deployment /DCFS monitoring panels configuration.md)

  Address：http://192.168.100.91:8086/

- Log Service： [es-kibana log service index configuration.md](./ steps of cold deployment/es- kibana log service index configuration.md)

  Address： http://192.168.100.152:5601/app/home#/

- Disk-Fault Background：Monitoring abnormal Disk read/write data, automatically deploy the service by bundling machine

  Address：http://192.168.100.152:3721/index.html

Monitor Service Status Monitoring: Robot warning

Current monitoring service list:

- etcd cluster
- disk_map_master

4. DMC MINER Deployment

4.1 Bind OOD

For Android

1、Install Cyber Chat.

2、Execute /cyfs/ood-installer –bind on the vps to obtain QR code for binding.

3、Use cyber chat to scan and bind OOD.

For IOS

1、 Install TestFlight on your device.

2、 Open the link https://testflight.apple.com/join/cDS8j9Ts and install Cyber Chat with TestFlight.

3、 Register an account of Cyber Chat.

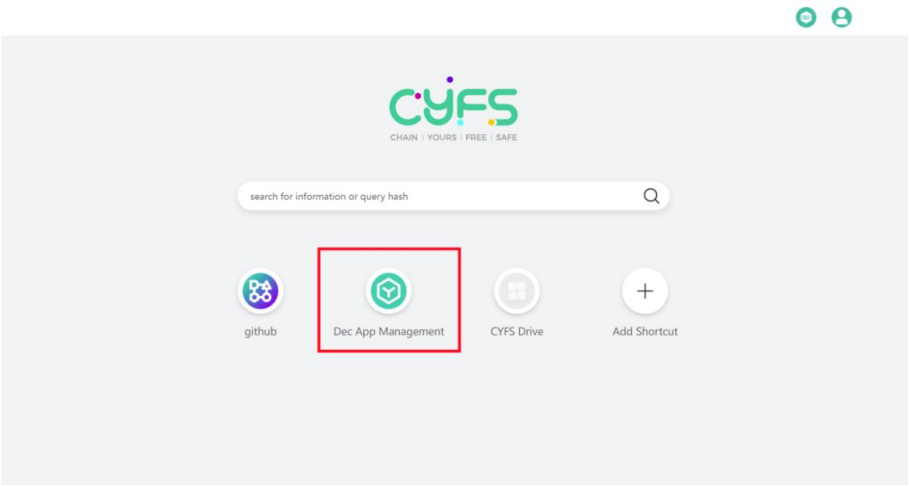
4、Use the client of Cyber Chat to scan and bind OOD.

4.2 Deploy CYFS Browser

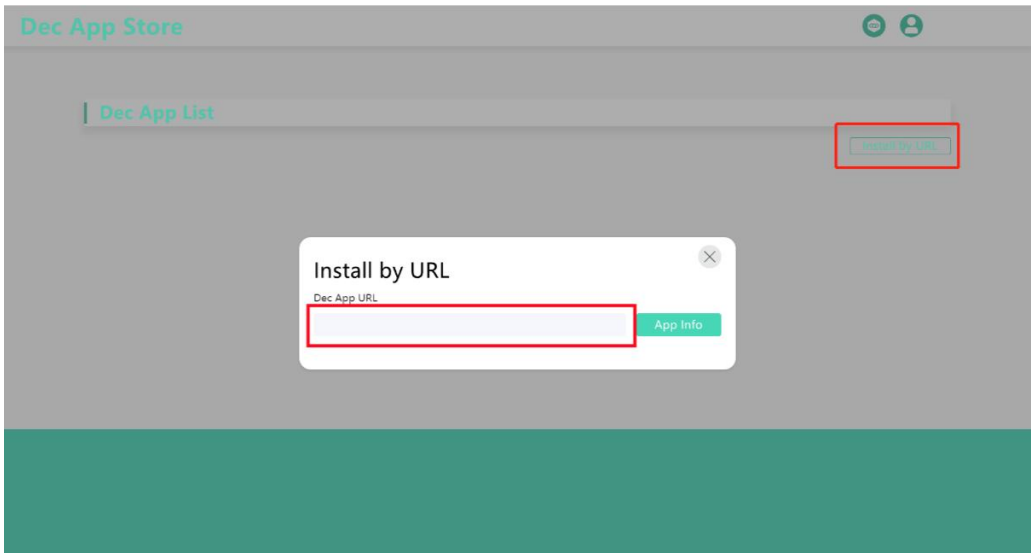
Steps of Installation

1. Insrall Cyfs Browser (Go to the DMC Github to download from cyfs-browser repositories)，then start the software and rgister (to bind OOD account).

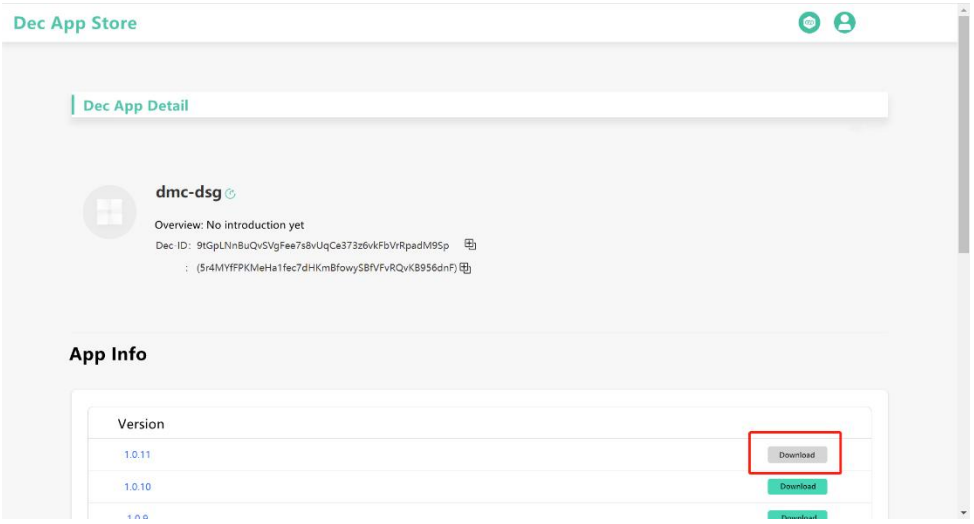
2. Click APP Management and open the page of APP Management.



3. Click “ install by URL” and input cyfs://5r4MYfFPKMeHa1fec7dHKmBfowySBfVFvRQvKB956dnF/9tGpLNnBuQvSVgFee7s8vUqCe373z6vkFbVrRpadM9S, then click to get the information of the Application



4. Choose the latest version of the dmc-dsg to install.



4.3 Install and Deploy DSG Miner

Miner command line interface:

Use dsg-miner-client.exe to execute the files and operate DMC account

Use method dsg-miner-client -h

Steps to use command line client:

1. Create a low-authority private key required for DSG operation, this command will return to the created light private key.

2. Set the DMC account and light private key used by DSG.

5. The Summary of Miner's Operation

Prerequisite introduction

-Token introduction 0,PST@datamall, indicates that this is a 0 decimal place token named PST issued by datamall. This will not be repeated in the following document.

-The code of this interface document takes JavaScript-SDK as an example, and it's same for other languages SDK.

-This document assumes that miner the provider (MP) has a sufficient amount of 4,DMC@datamall.

-This document is limited to the performance the process and the interface. Check the Technical Yellowpaper for more details.

- The following dmc examples are from

```
let dmc = DMC ( {
    chainId: config.chainId,
    keyProvider: config.keyProvider,
    httpEndpoint: config.httpEndpoint,
    logger: {
        log: null,
        error: null
    }
});
```

Query user's balance

Call

```
dmc.getTableRowsSync(true, "eosio.token", "dmc", "accounts").rows
```

Return

```
[
  {
    "primary": 0,
    "balance": {
      "quantity": "100000000.0000 DMC",
      "contract": "datamall"
    }
  }
]
```


Introduction

The above querying shows that dmc MP has 100000000.0000 DMC.

Create a new maker contract

The user who owns DMC can increase reserve to create maker contract to mint 0,PST@datamall.

Method Name

increase

Parameters

name	type	description
owner	string	miner \ limited partner
asset	string	the amount of staked DMC
miner	string	name of miner

Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.increaseSync("dmc", "100.0000 DMC@datamall", "dmc", {
    authorization: "dmc"
})
```

Introduction

- In the above code, dmc user increases reserve in dmc maker contract. The amount of staked DMC @datamall is 1.0000.

- If there is no corresponding maker contract to be created, then the smart contract will automatically create the maker contract.

Query the details of maker contract

Now we have created maker contract, we can query the details of maker contract.

Query

```
dmc.getTableRowsSync(true, "eosio.token", "eosio.token", "dmcmaker");
```

Return


```
[
  {
    "miner": "dmc",
    "current_rate": "1.25",
    "miner_rate": "1.00000000000000000",
    "total_weight": "10000.00000000000000000",
    "total_staked": {
      "quantity": "100.0000 DMC",
```

```
        "contract": "datamall"
    }
  }
]
```

Introduction

-This method can query all the maker contacts on the network.

- miner_rate indicates the miner's minimum stake proportion. Currently the stake proportion is 100%, indicating that the miner does not accept investments by others.

- current_rate indicates the stake rate of the current contract, check the Technical Yellowpaper for calculation details.

- total_weight indicates the total weight of the contract, and the initial total weight is 10000.

- total_staked indicates the total amount of DMC invested in the contract.

Modify the minimum stake rate of the miner

Currently, miners create a maker contract, but since the miners' stake rate is 100%, they cannot accept investments from other investors, but they can accept investments from other investors by setting the miners' minimum stake rate.

Method Name

setmakerrate

Parameters

name	type	description
owner	string	miner
rate	double	miner's minimum stake rate

Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.setmakerrateSync("dmc", 0.8, {
    authorization: "dmc"
})
```

Introduction

- The range is $0.2\leq r \leq1$, otherwise an error will be reported.

- Set the minimum stake rate of the miner in the maker contract of the dmc user to be 0.8, which means it will allow other investors to have a 20% share of the total share.

Invest maker contract

Users can choose to use 4,DMC@datmall to invest maker contract to get dividend.

Interface is same as increase.

Example

```
ctx.increaseSync("lp", "100.0000 DMC@datamall", "dmc", {
    authorization: "lp"
})
```

Introduction

- lp limited partner stakes 100.0000 DMC@datamall in the maker contract of dmc user.

Query

```
dmc.getTableRowsSync(true, "eosio.token", "eosio.token", "dmcmaker");
```

Return

```
{
    "miner": "dmc",
    "current_rate": "1.25",
    "miner_rate": "0.80000000000000001",
    "total_weight": "20000.00000000000000000",
    "total_staked": {
        "quantity": "200.0000 DMC",
        "contract": "datamall"
    }
```


Query

Query detailed proportion of investment

```
dmc.getTableRowsSync(true, "eosio.token", "dmc", "makerpool");
```

Return


```
[
    {
      "owner": "dmc",
      "weight": "10000.00000000000000000"
    },
   {
     "owner": "lp",
     "weight": "10000.00000000000000000"
   }
]
```


Introduction

- dmc and lp each have a weight of 10000, so the total weight is 20000.

- Of the subsequent dividend, dmc and lp will receive 50% of the dividend respectively.

Claim reserve

Reserve can be claimed

Method Name

redemption

Parameters

name	type	description
owner	string	miner \ limited partner
asset	string	the amount of staked DMC
miner	string	name of miner

Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.redemptionSync("dmc", 0.2, "dmc", {
    authorization: "dmc"
})
```

Introduction

- In the above code, dmc user claimed the reserve from the maker contract the miner for dmc user, among which the claim DMC proportion is 0.2, the contract will distinguish between miners and limited partners depending on the caller.

- You can claim the amount of DMC based on your proportion, if you want to receive the whole

amount, set the rate as 1.

Mint PST

Introduction: Minting PST has a lot of limitations, assuming that PST can be minted through the following operations.

Method Name

mint

Parameters

name	type	description
owner	string	miner
asset	string	the amount of minted PST

Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.mintSync("dmc", "100 PST@datamall", {
    authorization: "dmc"
})
```

Query

```
dmc.getTableRowsSync(true, "eosio.token", "dmc", "accounts").rows
```

Return

```
[
```

```
  ...
  ,
  {
    "primary": 1,
    "balance": {
      "quantity": "100 PST",
      "contract": "datamall"
    }
  }
]
```

As you can see, the user minted 100 PST@datamall successfully.

Pending PST

After obtaining PST，the user can trade PST for DMC.

Method Name

bill

Parameters

name	type	description
owner	string	account name of pending order
asset	string	amount of pending order
price	double	the price of pending order
memo	string	memo

Example

```
let owner = "testnetbppa1";
let price = 1;
```

```
let asset = "100 PST@datamall";
let memo = "test";
let ctx = dmc.contractSync("eosio.token");
ctx.billSync(owner, asset, price, memo, {
        authorization: owner
});
```

In above code, the price of pending order is 100 PST.

Notice: In order to distinguish the interface pending orders and the front-end system pending orders and to meet the needs of the interface memo of pending orders not displaying in the front-end, we have temporarily agreed that the front-end memo contents of pending orders are automatically prefixed with "input_memo:" at the beginning, memo that does not begin with “input_memo” will not be displayed in the front-end.

For example: "memo": "input_memo: Service expires on 2022-12-12", memo displayed in the front-end will be "Service expires on 2022-12-12"; "memo": "Service expires on 2022-12-12", the front-end memo will not be displayed.

Query

```
dmc.getTableRowsSync(true, "eosio.token", "dmc", "stakerec")
```

Return


```
{
  "primary": 0,
  "bill_id": "10448685463539245185",
  "owner": "dmc",
  "matched": {
    "quantity": "0 PST",
    "contract": "datamall"
  },
```

```
  "unmatched": {
     "quantity": "100 PST",
     "contract": "datamall"
    },
    "price": "4294967296",
  "created_at": "2022-07-18T13:39:49",
    "updated_at": "2022-07-18T13:39:49"
}
```


Introduction

- bill_id is the only ID of the pending order on the chain.

- matched is the amount of matched DMC of this order.

- unmatched is the amount of unmatched DMC of this order.

- price is the price of pending order, the actual price is $price_{real}=\frac{price}{2^{32}} $.

Collect the reward of pending order

The owner of pending order can collect the reward of unmatched pending orders.

Method Name

getincentive

Parameters

name	type	description
owner	string	collector
bill_id	int	order ID of pending order

Example

```
let name = "dmc";

let bill_id = "10448685463539245185";

let ctx = dmc.contractSync("eosio.token");

ctx.getincentiveSync(name, bill_id, {

        authorization: name

})
```

Introduction

- The reward is 8,RSI@datamall.

Swap RSI <-> DMC

Since the miner already has RSI and DMC now the miner can swap RSI and DMC on DMCswap.

Method Name

exchange

Parameters

name	type	description
owner	string	account name of the swapping
quantity	string	amount of swapping
to	string	the token type to be swapped for
price	double	price, must be 0
id	string	provider ID
memo	string	memo

Example

```
let owner = "dmc";

let quantity = "1000000.0000 DMC@datamall";

let to = "0.00000000 RSI@datamall";

let price = 0;

let id = "test";

let memo = "test";

let ctx = dmc.contractSync("eosio.token");

ctx.exchangeSync(owner, quantity, to, price, id, memo, {

        authorization: owner

});
```

- The above operation is to swap 1000000.0000 DMC@datamall for 8,RSI@datamall in DMCswap.

- And 8,RSI@datamall can be swapped for 4,DMC@datamall.

- 0.3% handling fee will be charged for trading, which will be deducted from the to.

Matching order

User can use 4,DMC@datamall to purchase 0,PST@datamall through a pending order contract, which means to purchase storage capacity. The purchased 0,PST@datamall will be entered into a storage delivery contract.

Method Name

order

Parameters

name	type	description
owner	string	account name who executes trading
miner	string	miner who sends pending order
bill_id	string	the order ID of pending order
asset	string	the amount of asset needed for transaction
reserve	string	pre-deposit amount of asset(could be 0
memo	string	memo

Example

```
let ctx = dmc.contractSync("eosio.token");

ctx.orderSync({

    "owner": user,

    "miner": miner,

    "bill_id": bill_id,

    "asset": "30 PST@datamall",

    "reserve": "0.0000 DMC@datamall",

    "memo": "test"

}, {

    authorization: user

})
```

Introduction

In the above code, dmc user can trade those orders with bill_id from the pending order table of the miner and 4,DMC@datamll will be deducted based on the price (price) of pending orders that are

corresponding to the order ID of asset*.

Upload and Download Files

sync_file synchronizing files, input the file path that needs to be synchronized

dmc-dsg-client sync_file –h

Prerequisite Introduction

- Since there are multiple methods that need to use the same data during the storage challenge, they are placed here for subsequent use.

```
const eosjs_ecc = require("eosjs-ecc");
let data = [];
let merkle_tree = [];
for (let i = 0; i < 4; i++) {
    data.push("lala" + i * i);
}
let hash_array = [];
for (let i = 0; i < data.length; i++) {
    hash_array.push(eosjs_ecc.sha256(data[i]));
}
merkle_tree.push(hash_array);
for (let i = 0; i < merkle_tree.length; i++) {
    let merkle_level = merkle_tree[i];
    let upper_nodes = [];
    if (merkle_level.length == 1) {
        break;
    }
```

```
    for (let j = 0; j < merkle_level.length; j += 2) {
        var tmp = Buffer.from(merkle_level[j], "hex");
        tmp.append(Buffer.from(merkle_level[j + 1], "hex"));
        upper_nodes.push(eosjs_ecc.sha256(tmp))
    }
    merkle_tree.push(upper_nodes);
}
```

Submitting Merkle roots

After the order is generated, both parties need to submit the same Merkle roots and reach a consensus before the order will go into delivery status.

Method Name

```
addmerkle
```

Parameters

parameter name	parameter type	memo
sender	name	executer
order_id	uint64_t	order ID
merkle_root	checksum256	merkel root hash
data_block_count	uint64_t	number of data block

Example

```
let ctx = dmc.contractSync("eosio.token");
ctx.addmerkleSync({
  sender: "wugren123453",
  order_id: "2490429520303063759",
```

```
  merkle_root: merkle_tree[merkle_tree.length - 1][0],
}, {
  authorization: "wugren123453"
});
```

Introduction

During arbitration， since the contract needs to generate Merkle tree leaf nodes based on the data, the rules for Merkle tree generation need to be specified.

1. Chunking data to a certain size.
2. Numbering the data, starting from 0.
3. Hash the **block of data** and form a Merkle tree.

- In the contract, the father node is generated by hashing the child nodes after merging their char buffer. Please be careful to use the same rules when generating the Merkle tree locally, otherwise it will cause problems with arbitration.

- After submitting the root of the consistent Merkle tree, the order automatically enters the delivery state.

Query

```
dmc.getTableRowsSync({
    json: true,
    code: "eosio.token",
    scope: "eosio.token",
    table: "dmchallenge",
    lower_bound: "2490429520303063759",
    upper_bound: "2490429520303063759",
    limit: 100
});
```

Return

- When only users or miners submit Merkle roots:


```
{
    "order_id": "2490429520303063759",
    "pre_merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "pre_data_block_count": 4,
    "merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "data_block_count": 0,
    "merkle_submitter": "wugren123453",
    "data_id": 0,
    "hash_data":
"0000000000000000000000000000000000000000000000000000000000000000",
    "challenge_times": 0,
    "nonce": "",
    "state": 0,
    "user_lock": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "miner_pay": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "challenge_date": "1970-01-01T00:00:00"
```

```
}
```


- When both sides submit consistent Merkle roots:


```
{
    "order_id": "2490429520303063759",
    "pre_merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "pre_data_block_count": 0,
    "merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "data_block_count": 4,
    "merkle_submitter": "eosio.token",
    "data_id": 0,
    "hash_data":
"0000000000000000000000000000000000000000000000000000000000000000",
    "challenge_times": 0,
    "nonce": "",
    "state": 1,
    "user_lock": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
      },
      "miner_pay": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
      },
      "challenge_date": "1970-01-01T00:00:00"
}
```

- state the introduction of state:

```
enum e_challenge_state {
    ChallengePrepare = 0,
    ChallengeConsistent = 1,
    ChallengeCancel = 2,
    ChallengeRequest = 3,
    ChallengeAnswer = 4,
    ChallengeArbitrationMinerPay = 5,
    ChallengeArbitrationUserPay = 6,
    ChallengeTimeout = 7,
};
```

Collect reward of delivery

Receive delivery rewards when one delivery cycle has ended.

Method Name

```
claimorder
```

Parameters

Parameter name	parameter type	memo
payer	string	executer
order_id	uint64_t	order ID

Introduction

- Orders must have been delivered for at least one delivery cycle before this method can be called.

- This method will only be called successfully once in a delivery cycle.

- The order status will not be real-time updated, there are cases when there is actually a reward available but the order status is not updated, you can try to collect it directly after one delivery

cycle.

- 80% of the final collected rewards go to the miner's account and 20% to the Stake pool. The miner's portion of the handling charge incurred during the storage challenge will also be deducted.

Example

```
let ctx = dmc.contractSync("eosio.token");

ctx.claimorderSync({

  "payer": "miner",

  "order_id": 0,

}, {

  "authorization": "miner"

});
```

Query

- All the miner’s order information

```
dmc.getTableRowsSync({

    json: true,

    code: "eosio.token",

    scope: "eosio.token",

    table: "dmcorder",

    key_type: "name",

    index_position: "tertiary",

    lower_bound: "miner",

    upper_bound: "miner",

    limit: 100

});
```

Result


```
{
   "order_id": "4845120205916081",
   "user": "chenyixun123",
   "miner": "zxcvbnm12345",
   "bill_id": "7646877604673459130",
   "user_pledge": {
         "quantity": "0.0000 DMC",
         "contract": "datamall"
    },
    "miner_pledge": {
         "quantity": "10 PST",
         "contract": "datamall"
    },
    "price": {
         "quantity": "0.1100 DMC",
         "contract": "datamall"
    },
    "settlement_pledge": {
          "quantity": "0.1100 DMC",
          "contract": "datamall"
    },
    "lock_pledge": {
          "quantity": "0.1100 DMC",
          "contract": "datamall"
    },
    "state": 1,
```

```
    "deliver_start_date": "2022-07-13T06:33:54",
    "latest_settlement_date": "2022-07-20T06:33:54"
}
```

- Introduction of order status:

```
enum e_order_state {
    OrderStateWaiting = 0,
    OrderStateDeliver = 1,
    OrderStatePreEnd = 2,
    OrderStatePreCont = 3,
    OrderStateEnd = 4,
};
```

Renew service circle of order（Optional）

The user pre-deposits DMC to an order for renewal.

Method Name

addordasset

Parameters

parameter name	parameter type	memo
sender	string	executer
order_id	uint64_t	order ID
quantity	string	amount of renewal cost

Introduction

- This method is only allowed to be called by the user of the order.

- It can be called at any time to reduce the cost of renewing the service, before and after the consensus is reached, in which case the contract will only deduct the user's pre-deposited DMC of the order. And other actions will not be occurred.

- When one service cycle has passed 6/7, the contract will estimate if the user has sufficient assets to renew the service. If not, the order will end when the service cycle completes.

Example

```
let ctx = dmc.contractSync("eosio.token");

 ctx.addordassetSync({

    "sender": "user",

    "order_id": "48451202059160 81",

    "quantity": "90.0000 DMC@datamall",

}, {

    authorization: "user"

})
```

Query

-Data of a random order

```
dmc.getTableRowsSync({

   json: true,

   code: "eosio.token",

   scope: "eosio.token",

   table: "dmcorder",

   lower_bound: "48451202059160 81",

   upper_bound: "48451202059160 81",

   limit: 100

});
```

Result


```
{
    "order_id": "4845120205916081",
    "user": "chenyixun123",
    "miner": "zxcvbnm12345",
    "bill_id": "7646877604673459130",
    "user_pledge": {
          "quantity": "0.0000 DMC",
          "contract": "datamall"
     },
     "miner_pledge": {
          "quantity": "10 PST",
          "contract": "datamall"
     },
     "price": {
          "quantity": "0.1100 DMC",
          "contract": "datamall"
     },
     "settlement_pledge": {
           "quantity": "0.1100 DMC",
           "contract": "datamall"
     },
     "lock_pledge": {
           "quantity": "0.1100 DMC",
           "contract": "datamall"
     },
     "state": 1,
```

```
    "deliver_start_date": "2022-07-13T06:33:54",
    "latest_settlement_date": "2022-07-20T06:33:54"
}
```

Reduce the service circle of order （Optional)

The user withdraws DMC to an order for reducing service cycle.

Method Name

```
subordasset
```

Parameters

parameters name	parameters type	memo
sender	string	executer
order_id	uint64_t	order ID
quantity	string	amount of renewal cost

Introduction

- This method is only allowed to be called by the user of the order.

- It can be called at any time to reduce the cost of renewing the service, before and after the consensus is reached, in which case the contract will only deduct the user's pre-deposited asset of the order. And other actions will not be occurred.

Example

```
let ctx = dmc.contractSync("eosio.token");
  ctx.subordassetSync({
    "sender": "user",
    "order_id": "48451202059160 81",
    "quantity": "90.0000 DMC@datamall",
}, {
```

```
    authorization: "user"
})
```

Query

The query method is same as renew service circle of order.

Initiate the storage challenge(optional)

The users can initiate storage challenge. They can choose to submit data block hash or submit null data hash.

Method Name

```
reqchallenge
```

Parameters

parameter name	parameter type	Memo
sender	name	user’s account
order_id	uint64_t	order ID
data_id	uint64_t	id of data block
hash_data	checksum256	hash(hash(data+nonce))
nonce	string	random obfuscated value

Example

```
const eosjs_ecc = require("eosjs-ecc");


let nonce = "nonce";

let pre_data_hash = eosjs_ecc.sha256(data[2] + nonce);

let data_hash = eosjs_ecc.sha256(Buffer.from(pre_data_hash, "hex"));

let ctx = dmc.contractSync("eosio.token");
```

```
ctx.reqchallengeSync({
  sender: "user",
  order_id: "2490429520303063759",
  data_id: 2,
  hash_data: data_hash,
  nonce: nonce,
}, {
  authorization: user
});
```

Introduction

- The user can choose the storage challenge with data, in which case the user needs to generate a random obfuscated string, and then hash (random data block + random string) and submit the data block, the block number, the obfuscated string and the result of the second hash to the blockchain.

- If the user chooses the storage challenge without data, the user can submit any character with a secondary hash and submit the block number that requires the miner's arbitration response, but note that the miner's response to the challenge must fail at this point, and the user will be judged as the party at fault after a successful arbitration and will have to pay a higher handling charge.

- The challenge handling charges will be deducted from DMC deposited by the user for the order, so user needs to ensure enough pre-deposit, otherwise userwill not be able to challenge.

Query

```
dmc.getTableRowsSync({
   json: true,
   code: "eosio.token",
   scope: "eosio.token",
   table: "dmchallenge",
   lower_bound: "2490429520303063759",
   upper_bound: "2490429520303063759",
   limit: 100
```

```
});
```

Return


```
{
    "order_id": "2490429520303063759",
    "pre_merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "pre_data_block_count": 0,
    "merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "data_block_count": 4,
    "merkle_submitter": "eosio.token",
    "data_id": 2,
    "hash_data": "922ca76218250def6a2cb78438d834228b7eaede11c60cd6140c7ccc7e27324b",
    "challenge_times": 1,
    "nonce": "nonce",
    "state": 3,
    "user_lock": {
          "quantity": "0.0110 DMC",
          "contract": "datamall"
     },
     "miner_pay": {
          "quantity": "0.0000 DMC",
          "contract": "datamall"
     },
     "challenge_date": "2022-07-14T11:52:33"
}
```


Respond to the challenge(optional)

When the user initiates a challenge, the miner can choose to respond to the challenge, and the miner needs to complete the response within a specified time (24 hours). The miner gets the block number and obfuscated string submitted by the user from the blockchain system, and subsequently hashes (the block number corresponding to the number + obfuscated string) and submits it to the blockchain.

Method Name

```
anschallenge
```

Parameters

parameter name	parameter type	memo
sender	name	account of Miner
order_id	uint64_t	order ID
reply_hash	checksum256	hash(data+nonce)

Introduction

- When the contract verifies that the miner submitted an invalid hash, the contract will throw an abnormality and will not upload this transaction to the chain. Therefore, there will a timeout violation of the challenge, an intermediate state will not be occurred that shows the challenge fails.

- When the miner submits a legitimate hash and the contract validates it, both parties will pay handling charge of 10% of the PST unit price in DMC.

- The miner's handling charge is settled when the `DMC` reward is claimed. The user's excess handling charge is deducted after the miner responses the challenge.

Example

```
const eosjs_ecc = require("eosjs-ecc");


let nonce = "nonce";

let reply_hash = eosjs_ecc.sha256(data[2] + nonce);

let ctx = dmc.contractSync("eosio.token");


ctx.anschallengeSync({
```

```
    sender: "miner",
    order_id: "2490429520303063759",
    reply_hash: reply_hash,
}, {
    authorization: "miner"
});
```

Query

```
dmc.getTableRowsSync({
    json: true,
    code: "eosio.token",
    scope: "eosio.token",
    table: "dmchallenge",
    lower_bound: "2490429520303063759",
    upper_bound: "2490429520303063759",
    limit: 100
});
```

Return

```
{
    "order_id": "2490429520303063759",
    "pre_merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "pre_data_block_count": 0,
    "merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "data_block_count": 4,
    "merkle_submitter": "eosio.token",
    "data_id": 2,
    "hash_data": "922ca76218250def6a2cb78438d834228b7eaede11c60cd6140c7ccc7e27324b",
```

```
    "challenge_times": 1,
    "nonce": "nonce",
    "state": 4,
    "user_lock": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "miner_pay": {
        "quantity": "0.0011 DMC",
        "contract": "datamall"
     },
     "challenge_date": "2022-07-14T11:52:33"
}
```


Arbitration(optional)

When the user initiates a challenge, miners can skip responding to the challenge and directly arbitrate or arbitrate after failed to respond to the challenge.

Method Name

```
arbitration
```

Parameters

parameter name	parameter type	memo
sender	name	account of miner
order_id	uint64_t	order ID
data	std::vector<char>	the data corresponding to the data number

cut_merkle	std::vector<checksum256>	merkel tree pruning

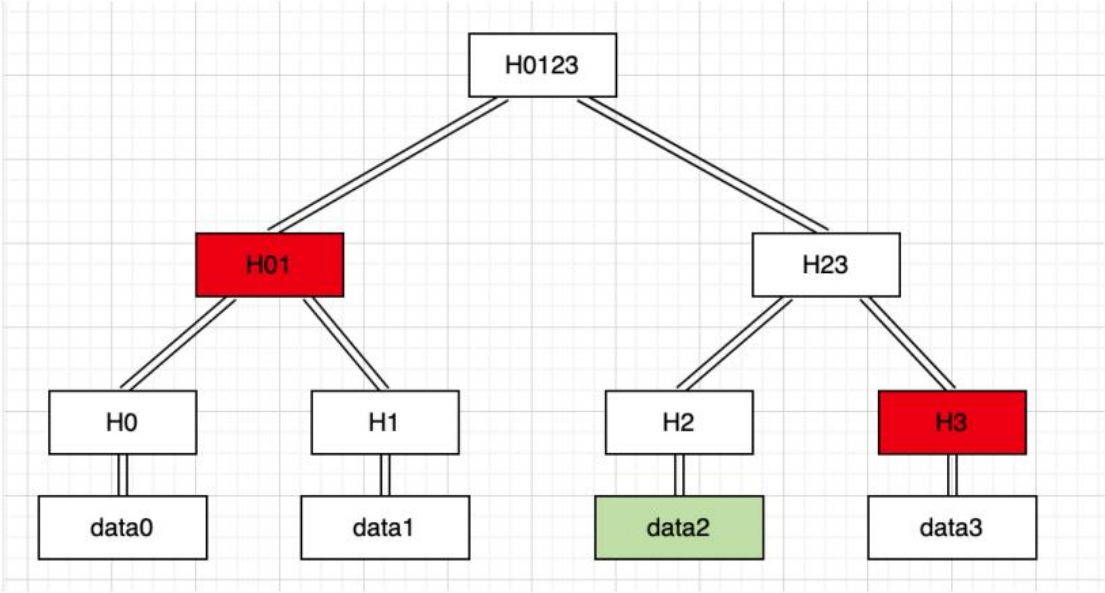
Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.arbitrationSync({
   sender: "miner",
   order_id: "2490429520303063759",
   data: Buffer.from(data[2]),
   cut_merkle: [
         merkle_tree[0][3],
         merkle_tree[1][0]
   ]
}, {
   authorization: "miner"
});
```

Introduction

For arbitration, the miner needs to submit the data corresponding to the data block number (in byte stream format), and a Merkle tree pruning corresponding to this data. The Merkle tree pruning only needs to be submitted to the brother node in order to save space. The figure shows that the data to be verified is in green and the Merkle tree root to be submitted is in red:



The organized way of Merkle trees pruning:

$$[H_3, H_{01}]$$

If the submitted data and the hash result of the Merkle tree pruning agree with the root of the Merkle tree submitted initially, the arbitration is successful. At this point the contract will hash (hash (submitted data block + stored obfuscated string)), if the calculation result is consistent with the hash uploaded by the user when initiating the challenge, the miner is the fault party; otherwise the user is the fault party, the fault party will pay a handling charge of 10 times the PST unit price in DMC, the non-fault party still has to pay a handling charge of 10% of the PST unit price in DMC.

When arbitration fails, the contract will throw an abnormality and the transaction will not be uploaded to the chain. If the challenge is not successfully responded or arbitration is not completed within the specified time (24 hours), the miner is deemed as having breached the contract and pays the DMC to the user.

Query

```
dmc.getTableRowsSync({
    json: true,
    code: "eosio.token",
    scope: "eosio.token",
```

```
    table: "dmchallenge",
    lower_bound: "2490429520303063759",
    upper_bound: "2490429520303063759",
    limit: 100
});
```

Return

```
{
    "order_id": "2490429520303063759",
    "pre_merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "pre_data_block_count": 0,
    "merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "data_block_count": 4,
    "merkle_submitter": "eosio.token",
    "data_id": 2,
    "hash_data": "922ca76218250def6a2cb78438d834228b7eaede11c60cd6140c7ccc7e27324b",
    "challenge_times": 1,
    "nonce": "nonce",
    "state": 5,
    "user_lock": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "miner_pay": {
        "quantity": "0.0110 DMC",
        "contract": "datamall"
     },
```

```
    "challenge_date": "2022-07-14T11:52:33"
}
```

Timeout and compensate (optional)

When the user initiates a challenge and the miner hasn't responded within a period of time (24 hours) or the response fails, a time-out compensate can be initiated and an amount of DMC for purchasing the PST of the order will be paid the user by the miner, and the order status will be set as time-out.

Method Name

```
paychallenge
```

Parameters

parameter name	parameter type	memo
sender	name	User's account
order_id	uint64_t	order ID

Example

```
let ctx = dmc.contractSync("eosio.token");


ctx.paychallengeSync({
  sender: "user",
    order_id: "2490429520303063759"
}, {
  authorization: "user"
});
```

Introduction

- Within the timeout and compensate process, no handling charge will be charged and the funds will be returned.

- The amount of compensation will be deducted from the miner's pool, 50% to the user and 50% to the DMC foundation.

Query

```
dmc.getTableRowsSync({
    json: true,
    code: "eosio.token",
    scope: "eosio.token",
    table: "dmchallenge",
    lower_bound: "2490429520303063759",
    upper_bound: "2490429520303063759",
    limit: 100
});
```

Return

```
{
    "order_id": "2490429520303063759",
```

```
    "pre_merkle_root":
"0000000000000000000000000000000000000000000000000000000000000000",
    "pre_data_block_count": 0,
    "merkle_root":
"82368064979867335ff915ae802dcbd51c24e9e42c938ee2e288aa08ad84d275",
    "data_block_count": 4,
    "merkle_submitter": "eosio.token",
    "data_id": 2,
    "hash_data":
"922ca76218250def6a2cb78438d834228b7eaede11c60cd6140c7ccc7e27324b",
    "challenge_times": 1,
    "nonce": "nonce",
    "state": 7,
    "user_lock": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "miner_pay": {
        "quantity": "0.0000 DMC",
        "contract": "datamall"
     },
     "challenge_date": "2022-07-14T11:52:33"
}
```

6. Analysis of the Revenue of Miners

Let's take the data of stage 2 from whitepaper as an example, the stage production is 142560000 DMC.

After calculation, we can conclude that the DMC release is 33000 DMC per hour.

Assuming that the total storage capacity on the DMC chain is 100 P at this time, our miner has 1 P of storage capacity, i.e. our miners accounts for 1% of the total storage capacity on the chain.

Then the daily revenue of miner 1P storage capacity is 33000 × 24 × 1% = 7920 DMC.