Tokens

Purpose of the cyber.token smart contract development

The system smart contract `cyber.token` provides token management functions, generates new tokens and stores information about created tokens and provides an ability to conduct mutual settlements between accounts as well.

The `cyber.token` smart contract includes the following actions: create, issue, retire, transfer, bulktransfer, payment, bulkpayment, claim, open and close.

The create action

The `create` action is applied to create a token to supply it into the system. This action has the following form:

```
[[eosio::action]] void create(
    name    issuer,
    asset   maximum_supply
);
```

Parameters:

- `issuer` — account name which creates the token to supply it into the system.
- `maximum_supply` — a structure value containing the fields:
  - maximum number of tokens supplied;
  - token symbol (data type that uniquely identifies the token). This is a structure value containing fields:
    - token name, consisting of a set of capital letters;
    - field that specifies a token cost accuracy in the form of decimal places number.

The `issuer` account is authorized to supply and withdraw tokens from circulation in the system. The `create` action can be executed by the `token` smart contract only. In order to perform this action, the signature of the validators is required. Use of the `bandwidth` resources (RAM) is charged to the `issuer` account.

The issue action

The `issue` action is applied to put a token into circulation in the system.
This action has the following form:

```
[[eosio::action]] void issue(
    name to,
    asset quantity,
    string memo
);
```

Parameters:

- `to` — recipient account to balance of which the tokens are transferred.
- `quantity` — number of the supplied tokens. This is a structure value containing fields:
    - number of the supplied tokens in the system;
    - a token symbol. This is a structure value containing fields:
        - the token name, consisting of a set of capital letters;
        - the field that specifies a token cost accuracy in the form of decimal places number.
- `memo` — memo text that clarifies a meaning (necessity) of the token emission in the system. The text volume should not exceed 384 symbols including blanks.

When the `create` action is performed, the token symbol and the account name `issuer` are put into the table. When executing the `issue` action, the token character is taken from the resulting `quantity` value. The account `issuer` can be determined via using this symbol and table data. The number of supplied tokens should not exceed `maximum_supply` value specified in the `create` action. Use of the `bandwidth` resources (RAM) is charged to the `issuer` account.

To perform the `create` action, the `issuer` account authorization is required.

The retire action

The `retire` action is used to withdraw a certain number of tokens from the system. This action has the following form:

```
[[eosio::action]] void retire(
    asset quantity,
    string memo
);
```

Parameters:

- `quantity` — number of withdrawn tokens.
- `memo` — a memo text clarifying a purpose of withdrawing tokens from circulation.

Use of the `bandwidth` resources (RAM) is charged to the `issuer` account. The number of tokens withdrawn from circulation is also removed from `issuer` account balance, so this account can not withdraw tokens more than he/she has them on own balance.

To perform this action, the `issuer` account authorization is required.

The transfer action

The `transfer` action is used to transfer tokens from one account to another. This action has the following form:

```
[[eosio::action]] void transfer(
    name from,
    name to,
    asset quantity,
    string  memo
);
```

Parameters:

- `from` — sender account from balance of which the tokens are withdrawn.
- `to` — recipient account to balance of which the tokens are transferred.
- `quantity` — number of tokens to be transferred. This value should be greater than zero.
- `memo` — a memo text that clarifies a meaning of the token transfer.

The action is performed with sending a notification to smart contracts of the sender and recipient of tokens. Sending the notification is an internal action and looks similar to the `transfer` action . The difference is the «sending a notification» action is not performed on `cyber.token`, but on smart contracts of sender and recipient of the notification (for example, if the vesting smart contract receives a notification, it automatically charges tokens on the `vesting` balance).

To perform the `transfer` action, the `from` account authorization is required.

Use of the `bandwidth` (RAM) resources is charged either to sending account or to receiving account, depending on who signed the transaction. If the `open` action was previously performed, none of them should pay `bandwidth`, since the record already created in database is used.

> **Note:**
> If sender account pays for used resources (RAM) for a recipient account, then there is the possibility of spending the entire share of memory allocated to the recipient account. To avoid such cases when performing the action `transfer`, additional actions have been implemented in `cyber.token`. These are `open` and `close`. Functional purposes of these actions are the creation and deletion of an entry in the database before and after (not during) performing `transfer` respectively.

The bulktransfer action

The action `bulktransfer` is used to transfer tokens from one account's balance to the balances of several accounts (for example, for paying rewards for a post). This action has the following form:

```
[[eosio::action]] void bulktransfer(
    name from,
    vector<recipient> recipients
)
```

Parameters:

- `from` — sender account from balance of which the tokens are withdrawn.
- `recipients` — array (list) of token recipients. Each array element is a structure containing fields:
  - `to` — recipient account to balance of which the tokens are transferred;
  - `quantity` — number of tokens to be transferred to account `to` . This value should be greater than zero;
  - `memo` — a memo text that clarifies a meaning of the token transfer. The text volume should not exceed 384 symbols including blanks.

The restrictions imposed on the `bulktransfer` action:

- transfer of various types tokens is not allowed;
- transfer of tokens to itself is not allowed, that is, the list of `recipients` should not contain the account name `from` .

It is allowed in the `recipients` list to specify the same account name `to` more than once. The action is performed with sending a notification to smart contracts of the sender and recipient of tokens in the same way as `transfer` .

To perform `bulktransfer` , it is required a signature of the account `from` .

The payment action

The `payment` action like the `transfer` action is used to transfer funds from one account to another. This action has the following form:

```
[[eosio::action]] void payment(
    name  from,
    name  to,
    const asset& quantity,
    const string& memo
)
```

Parameters:

- `from` — sender account from balance of which the tokens are withdrawn.
- `to` — account name which is a recipient token.
- `quantity` — number of tokens to be transferred to account name `to` . This value should be greater than zero.
- `memo` — a memo text that clarifies a meaning of the token transfer. The text volume should not exceed 384 symbols including blanks.

Unlike `transfer` , when performing `payment` action, notifications are not sent and funds are transferred not to the recipient account balance , but to the payment-intermediary balance. To withdraw funds from the `payment` balance, the recipient account `to` should additionally perform the action `claim` .

The action `payment` is a more secure option to transfer tokens. To perform `payment` , it is required a signature of the account `from` .

The bulkpayment action

The action `bulktransfer` is used to transfer tokens from one account's balance to several accounts (for example, for paying rewards to curators and beneficiaries for a post). The `bulkpayment` action has the following form:

```
[[eosio::action]] void bulkpayment(
    name from,
    vector<recipient> recipients
)
```

Parameters:

- `from` — sender account from balance of which the tokens are withdrawn.
- `recipients` — array (list) of token recipients. Each array element is a structure containing fields:
  - `to` — account name which is a recipient token;
  - `quantity` — number of tokens to be transferred to account name `to` . This value should be greater than zero;

    - `memo` — a memo text that clarifies a meaning of the token transfer. The text volume should not exceed 384 symbols including blanks.

The restrictions imposed on the `bulkpayment` action:

- transfer of various types tokens is not allowed;
- transfer of tokens to itself is not allowed, that is, the list of `recipients` should not contain the account name `from`.
  Unlike `bulktransfer`, when performing `bulkpayment` action, notifications are not sent and funds are transferred not to the recipient accounts balances , but to the payment (intermediary) balance. To withdraw funds from the `payment` balance, each of the recipient accounts should additionally perform the action `claim`.

The action `bulkpayment` is a more secure option to transfer tokens. To perform `bulkpayment`, it is required a signature of the account `from`.

The claim action

The `claim` action is used to transfer funds from the `payment` balance to an account balance. This action has the following form:

```
[[eosio::action]] void claim(
    name owner,
    asset quantity
)
```

Parameters:

- `owner` — account name to whose balance the funds are transferred.
- `quantity` — requested number of tokens to be transfered. This number must not exceed the number of tokens that are on the `payment` balance and which are owned by the `owner` account.

Performing the `claim` action requires a signature of the `owner` account.

The open action

The `open` action is used to create a record in database. This entry must contain an account name which should pay for the memory used, as well as a symbol for which this entry is created. The `open` action has the following form:

```
[[eosio::action]] void open(
    name owner,
    symbol symbol,
    name ram_payer
);
```

Parameters:

- `owner` — account name to which the memory is allocated.
- `symbol` — symbol for which the entry is being created.
- `ram_payer` — account name that pays for the used memory.

Performing the `open` action requires a signature of the `ram_payer` account.

The close action

The `close` action is an opposite of `open` and is used to free allocated memory in database. The `close` action has the following form:

```
[[eosio::action]] void close(
    name owner,
    symbol symbol
);
```

Parameters:

- `owner` — account name to which the memory was allocated.
- `symbol` — a symbol for which the entry is deleted.

To perform this action, it is necessary to have two zero balances of the account `owner` :

- zero token balance (determined by the symbol);
- zero payment balance.

Obtaining statistical information about system tokens

To obtain statistical information on tokens, two tables, `currency_stats` and `account` , are used in the `cyber.token` smart contract.

The `currency_stats` table has the following form:

```
struct [[eosio::table]] currency_stats {
    asset supply;
    asset max_supply;
    name issuer;
};
```

Parameters:

- `supply` — number of a certain type tokens that have been supplied in the system.
- `max_supply` — maximum possible number of this type tokens in circulation.
- `issuer` — account name which has supplied this type tokens in the system.

Primary key for table `currency_stats` is a character value in the `asset` . This value is used to determine the token. having information about a token it is easily to obtain its `supply` value — the number of tokens released into circulation, as well as the account name `issuer` .

The `account` table has the following form:

```
struct [[eosio::table]] account {
    asset balance;
    asset payments;
};
```

A scope of the table is determined by the account name.