Stake

The `cyber.stake` system smart contract provides voting methods for validators, which allow users to use their votes, as well as resources on balance sheets, most effectively. `Cyber.stake` also provides the user with the ability to authorize another user to vote.

Terminology used

Agent — is a potentially active user who has a balance in the `cyber.stake` smart contract.

Votes for a validator — a certain number of staked tokens given to the validator as votes during a voting (one token equals to one vote).

Delegator — a user who delegates part of his stake (RAM, CPU, NET, Storage resources) or an authorization to vote to another user during the voting process.

Proxy account — a user empowered with a specific voting authorization during the voting process. Any user can declare himself a proxy account in order to receive votes from other users and use these votes at his own discretion during the voting for validators. In that case the user must set the proxy account level by calling `setproxylvl` .

A user who does not participate in the voting for validators regardless of reason, can authorize the proxy account to vote instead of him. A proxy account can also empower another proxy account in order to vote on his behalf.

Proxy account level — a conditional division of users into categories. Each user category is assigned a specific proxy account level. The highest level of proxy account is zero, which is assigned only to validators. The first and further levels in ascending order are assigned to users who have declared themselves proxy accounts. The number of proxy accounts of the same level is also not limited. The last level of proxy account is assigned to the ordinary user. A proxy account can only be configured at only one of the following levels (**please note: in the current release, the number of proxy account levels is limited to three**):

- A validator is considered as a zero-level proxy account.

- First level proxy account — a user who has declared himself a proxy account, empowered with the votes of ordinary users at his disposal during the voting process.
- The second level proxy account is an ordinary user. The ordinary user does not have to worry about what level of proxy he/she needs to set. The user is automatically assigned to the lowest-rated proxy level.

The proxy accounts' separation by levels was introduced in order to avoid the isolation when transferring the voting right from one user to another (for example, user A can transfer his vote to user B, and user B can transmit the vote received to user C. If the latter transfers the received votes back to user A, it will lead to the abruption of transmissions. The concept of proxy account levels was introduced to exclude the appearance of isolation during the transfer of votes. Transfer of voting rights is open only to a lower level proxy account.

Stake — a share of bandwidth resources (RAM, NET, CPU, and Storage) allocated to a user. The user can manage the share of resources allocated to him both independently and entrust its use to another user (delegate the share of resources).

Staked tokens — the tokens allocated for a stake acquisition that can't be used for anything else in this state. The user can stake active tokens listed on his/her balance or deposit them. Also, the user can perform the reverse operation — withdraw tokens from the staked state to active.

---

cyber.stake smart contract actions

The following actions are assemble the `cyber.stake` smart contract: create, enable, open, delegatevote, setgrntterms, recallvote, withdraw, setproxylvl, setproxyfee, setminstaked, setkey, updatefunds, reward, pick, delegateuse, recalluse and claim. Additionally, the following inline functions can be found in this smart contract: get_votes_sum and get_top.

---

create

The create action is used to create a stake for a separate type (symbol) of tokens. After creating the stake, tokens of this type can be transferred by users to the state of the stake.

```
1  void create(
2     symbol token_symbol,
```

```
    std::vector<uint8_t> max_proxies,
    int64_t depriving_window,
    int64_t min_own_staked_for_election
)
```

Parameters:

- `token_symbol` — symbol of the token for which the stake is being created.
- `max_proxies` — maximum allowed proxy account values. The zero element of the vector sets the maximum number of users who can be trusted in the voting if the proxy account level is the first. The first element of the vector sets the maximum number of parts into which the stake can be divided if the level of the proxy account is second. If there are fourth or more levels in the proxy system, the vector will contain three or more elements.
- `depriving_window` — the length of the period (in seconds) during which funds withdrawn back to the delegate cannot be used by him. Funds can be recalled by `recallvote` operation call. Withdrawn funds during this period cannot be taken into account when calculating the share of resources.
- `min_own_staked_for_election` — the minimum number of tokens staked which the user must have in his/her repository to be a candidate for validators.

Generic users can vote for one validator either transfer the authorization for voting to only one proxy account.

A proxy account can simultaneously vote for several validators. The maximum number of validators for which the proxy account can vote is 30.

Calling create requires the rights of the creator of the `token_symbol` token. The token creator, by calling create, allows other users to translate tokens of this type into the state of the stake. When creating a stake for a system token, you must have system rights.

---

enable

The `enable` action allows to deduce a certain type of token from use for a while. This action sets the enable flag to true.

```
void enable(symbol token_symbol)
```

The `token_symbol` parameter is the symbol of the token affected by the enable flag set to true.

This action is used in testing to disable the use of tokens of a certain symbol. During testing, the enable flag is set to false, that is, the resource limit for a certain type of token is disabled. At the end of testing, the flag should be set back to true. If `token_symbol` is a system one, then when the enable flag is set on the node, the restriction on the use of resources by stake is enabled.

The rights of the `token_symbol` token creator are required to call this action.

---

open

The `open` action is used to create a balance for a user.

```
void open(
   name owner,
   symbol_code token_code,
   std::optional<name> ram_payer = std::nullopt
)
```

Parameters:

- `owner` — username for which balance is being created.
- `token_code` — symbol of the token that will be listed on the balance.
- `ram_payer` — name of the account paying for opening the balance. If the parameter accepts the default value `std::nullopt`, then the owner himself pays for opening the balance.

A transaction, containing the `open` action should be signed by the user who pays for opening a balance. Owner rights are being requested by default.

---

delegatevote

The `delegatevote` action is used to delegate to another account the right to dispose of staked tokens (in whole or in part) when voting for validators.

```
void delegatevote(
   name grantor_name,
   name recipient_name,
   asset quantity
)
```

Parameters:

- `grantor_name` — an account who delegates the right to dispose of a steak. This is either an ordinary user or a proxy account whose proxy level higher than that of a recipient.
- `recipient_name` — name of the proxy account that is trusted with the voting right.
- `quantity` — number of staked tokens that the proxy `recipient_name` can use when voting. The parameter must be greater than zero.

If a user for some reason can not vote for validators, he/she can use the services of a proxy account. In this case this user has to delegate to this proxy a certain number of staked tokens. These tokens (stake) can be used only when voting for validators.

Depending on the settings of the stake, the right to vote on the use of stake tokens can be transferred further (by calling `delegatevote`) to the ërecipient_nameë recipient, whose `proxy_level` proxy level is lower than that of the sender `grantor_name`.

> **Note:** Since in the current release the number of levels of proxy accounts is limited to three (zero is a validator, the first is directly a proxy account, the second is an ordinary user), voice transfer from one proxy account to another is not possible. Having received a vote from an ordinary user, a proxy account can use it only when voting for validators.

The stake is divided into several parts. The number of shares depends on the settings of the stake during the create action and is limited by the value of max_proxies. this value is equal to one for an ordinary user. Therefore the user can vote for one validator only.

A transaction, containing the `delegatevote` action should be signed by the `grantor_name` account.

---

recallvote

The `recallvote` action is used to withdraw the right to use a delegated stake when voting for validators. The steak value is set as a percentage and can be withdrawn either partially or completely.

```
void recallvote(
   name grantor_name,
   name recipient_name,
   symbol_code token_code,
   int16_t pct
)
```

Parameters:

- `grantor_name` — name of the account that withdraws the right to use the stake.
- `recipient_name` — name of the account from which the right to use the stake is withdrawn.
- `token_code` — staked token symbol.
- `pct` — share of stake (in percent), the right to use of which is revoked.

The recalled share of the stake can be used by the `grantor_name` account immediately after the `recallvote` operation is completed.

A transaction, containing the `recallvote` action should be signed by the `grantor_name` account.

---

setgrntterms

The `setgrntterms` action is used to distribute additionally allocated staked tokens between validators when voting. These tokens are distributed in accordance with the specified proportions..

```
void setgrntterms(
   name grantor_name,
   name recipient_name,
   symbol_code token_code,
   int16_t pct,
   int16_t break_fee,
```

```
    int64_t break_min_own_staked
)
```

Parameters:

- `grantor_name` — name of the account that delegates the extra share of stake.
- `recipient_name` — name of the proxy account, recipient of the extra share of staked tokens.
- `token_code` — staked token symbol.
- `pc` t — share (in percent) of additional votes received, which will be given for the `recipient_name` candidate.
- `break_fee` — the parameter that controls the change in the fee set by the candidate. The value of this parameter is the percentage of commission charged by the `grantor_name` account for the `recipient_name` candidate. If the candidate during the voting process sets the value of fee greater than `break_fee`, the recall operation will be automatically called to recall the votes of this candidate.
- `break_min_own_staked` — the parameter that controls the availability of secured tokens in the `recipient_name` candidate. The value of this parameter is set by the `grantor_name` account. By default, this change is set to the current `min_own_staked` value. If the candidate reduces `min_own_staked` (in order to withdraw funds to a liquid state), the next time the balance of the `grantor_name` account is updated, the recall operation will be automatically called to revoke the allocated tokens from this candidate.

When recalling, the condition is checked so that the candidate has funds in the amount of at least `min_own_staked`.

A proxy account can vote simultaneously for several candidates for validators, distributing votes between them in a certain proportion. The proxy account can receive an additional number of votes (staked tokens) and use them until the end of voting process. These additional votes will automatically be given to those candidates for whom the proxy account voted before receiving these votes. Additional votes will be distributed among the candidates taking into account previous proportion .

---

withdraw

The `withdraw` action is used to withdraw staked tokens and make them liquid.

```
void withdraw(
   name account,
   asset quantity
)
```

Parameters:

- `account` — name of the account withdrawing staked tokens.
- `quantity` — number of tokens withdrawn. This value shpild be positive.

When the `withdraw` action is calling, the `cyber.stake` smart contract sends a notification to token creator, which makes the final decision on the withdrawal of tokens. The withdrawal of tokens is performed immediately without any delay.

In case of withdrawal the system tokens, decision to withdraw these tokens is made by the `cyber.bios` smart contract. The operation is performed if the remainder of the user's stake covers the costs of the resources used by him taking into account the funds being withdrawn.

A transaction, containing the `withdraw` action should be signed by the `account` .

---

setproxylvl

The `setproxylvl` action sets the proxy account level for a user.

```
void setproxylvl(
   name account,
   symbol_code token_code,
   uint8_t level
)
```

Parameters:

- `account` — name of the account for which a proxy account level is set.
- `token_code` — number of staked tokens available to the user.
- `level` — level being set.

If a user intends to be a candidate for validators, he/she should set the proxy level to zero. If a user intends to register as a proxy account, he/she should set the proxy level to one (in the current release).

A transaction, containing the `setproxylvl` action should be signed by the `account`.

---

setproxyfee

The `setproxyfee` action sets a size of the commission for an agent (a candidate for validators) that users voted for.

```
void setproxyfee(
   name account,
   symbol_code token_code,
   int16_t fee
)
```

Parameters:

- `account` — name of the agent that users have voted for.
- `token_code` — symbol of tokens constituting a reward to the agent.
- `fee` — amount of commission deductions to be taken from the amount of reward that the agent receives. This parameter is calculated at the time of reward payment. The parameter takes values from «0» to «10000» inclusive («0» corresponds to «0 %», «10000» corresponds to «100 %»).

A fee is deducted from the received reward amount allocated to the agent and transferred to the agent account. The rest of the reward is distributed among all users who voted for this agent.

---

setminstaked

The `setminstaked` action sets the minimum possible size of a steak allocated for voting by an agent (a candidate for validators).

```
void setminstaked(
   name account,
   symbol_code token_code,
   int64_t min_own_staked
)
```

Parameters:

- `account` — name of the agent.
- `token_code` — symbol of staked tokens.
- `min_own_staked` — minimum possible steak, which is allocated for voting by the agent himself. This value should be positive.

If the candidate for validators sets the value `min_own_staked` less than `min_own_staked_for_election` one during a voting, then the votes cast for him are automatically canceled.

---

setkey

The `setkey` action sets the public key for a validator, which will be used for signing blocks.

```
void setkey(
   name account,
   symbol_code token_code,
   public_key signing_key
)
```

Parameters:

- `account` — name of the validator to which the public key is being set.
- `token_code` — symbol of staked tokens.
- `signing_key` — the public key (i.e. validator signature).

---

updatefunds

The `updatefunds` action is used to update a user's stake.

```
void updatefunds(
   name account,
   symbol_code token_code
)
```

Parameters:

- `account` — name of the account for which the steak size is being updated.
- `token_code` — symbol of staked tokens.

The size of one user's stake may vary. The `updatefunds` action sets the actual value for a user's stake. This operation simplifies the procedure for calculating user rewards, including share determination for a user. The `updatefunds` action can also be used during other actions, such as revoking tokens or voting. That is, when performing active operations that require updating a stake.

The `updatefunds` action can be called by any user. Authorization is not required.

---

reward

The `reward` action is used to pay rewards to users by crediting the corresponding amounts to their balances.

```
void reward(
   std::vector<std::pair<name, int64_t> > rewards,
   symbol sym
)
```

Parameters:

- `rewards` — a vector containing account names and the payouts associated to them.
- `sym` — symbol of tokens in which the reward is being paid.

A transaction, containing the `reward` action should be signed by the `sym` token creator.

---

pick

The `pick` action updates the entire list of validators, including candidates for validators, and notifies the `cyber.stake` contract about changes in the list, as well as about appearance of a new validator selected from the candidates at a certain time.

```
void pick(
   symbol_code token_code,
   std::vector<name> accounts
)
```

Parameters:

- `token_code` — number of tokens held by a candidate, notification of which is sent.
- `accounts` — vector of accounts that are current validators, as well as candidates for validators.

Selection of validators from the candidates for validators is carried out in the `cyber.govern` smart contract. The first `n-1` validators from the list (`n` is total number of validators) are selected in accordance with the number of votes cast for them, taking into account the weight of each vote. The last validator from this list is pseudo-random. This validator is selected according to a formula that takes into account the time elapsed since the last validator selection and the number of votes cast for this candidate.

Those candidates who did not become validators, but are listed next to the first `n-1` validators, get the opportunity to appear among these validators after a certain time. Candidate priorities are calculated using `get_top` method in the `cyber.govern` smart contract. This method determines the selected candidates who received the most votes and the reserve candidates. The method takes into account the length of stay in the status of a candidate (since when did the candidate remain among the non-elected). Each of the candidates is located in the list of the priority field in accordance with the priority calculated for it.

The `pick` operation is called by the `cyber.govern` smart contract and updates the priority field, moving all candidates in the queue to positions according to their calculated priority, and

notifies `cyber.stake` of changes in the candidate list at the current time.

A transaction, containing the `pick` action should be signed by the `token_code` token creator

---

delegateuse

The `delegateuse` action is used to delegate resources (RAM, NET, CPU, Storage) to another user, which can be used at user's discretion (unlike this operation, `delegatevote` delegates resources that can be used only for voting).

```
void delegateuse(
   name grantor_name,
   name recipient_name,
   asset quantity
)
```

Parameters:

- `grantor_name` — name of the account, who is a delegator of a stake.
- `recipient_name` — name of the account, who is a recipient of a stake.
- `quantity` — amount of delegated stake. This parameter should be positive.

Stake resources such as RAM, NET, CPU, and Storage cannot be delegated in parts as they appear on the system. Therefore, the delegation of stake funds is carried out in value terms. The cost of resources consumed by the delegator is calculated in accordance with current prices and compared with his steak existing.

The delegated steak is not taken into account when votes for the delegator are calculated, but is taken into account in the calculation of votes for the recipient of these funds.

When the `delegateuse` action is calling, the `cyber.stake` smart contract sends a notification to the token creator, who makes the final decision on the stake delegation.

In case of delegating the system tokens, decision to delegate these tokens is made by the `cyber.bios` smart contract. The operation is performed if the remainder of the delegator's

stake covers the costs of the resources used by him taking into account the funds being delegated.

---

recalluse

The `recalluse` action is used to revoke the delegated portion of a stake.

```
void recalluse(
   name grantor_name,
   name recipient_name,
   asset quantity
)
```

Parameters:

- `grantor_name` — name of the account revoking the delegated portion of the stake.
- `recipient_name` — name of the account whose delegated portion of the stake is being revoked.
- `quantity` — amount of revoked stake. This parameter should be positive.

As soon as the delegated part of the steak is withdrawn from the recipient, it immediately goes into a «frozen» state. The duration of the «frozen» state is determined by the `depriving_window` parameter when the create operation is called. The delegator can use the returned portion of the stake only after this period.

In case, after delegating the funds, only half of these funds are requested back, and then the remaining part is requested, then after the second `recalluse` call, the returned funds can only be used after the `depriving_window` time has passed.

---

claim

The claim action is used to obtain permission to use revoked delegated funds.

```
void claim(
   name grantor_name,
   name recipient_name,
   symbol_code token_code
)
```

Parameters:

- `grantor_name` — name of the account that revokes the delegated portion of the stake
- `recipient_name` — name of the account whose delegated portion of the stake is being revoked.
- `token_code` — revoked part of the stake. Value should be positive.

After the expiration of the «frozen» state, the withdrawn funds may not be used by the delegator immediately, but only after obtaining the appropriate permission. The claim call requests permission from the cyber.stake smart contract to use funds withdrawn from the «frozen» state.

The operation is tied to resource recovery time. Funds are immediately returned in full.

The delegator can again return the withdrawn funds (or part of the funds) back to the recipient_name account. The funds for redelegation will be debited from the «frozen» funds.

To perform the action, authorization of the funds delegator is required.

---

get_votes_sum

The `get_votes_sum` inline function returns the total number of votes available to validator candidates who received the most votes by validator candidates who received the most votes.

```
static inline int64_t get_votes_sum(
   symbol_code token_code,
   size_t limit = 0
)
```

Parameters:

- `token_code` — symbol of staked tokens by which votes were determined.
- `limit` — the requested number of first candidates from the priority list for which the total number of votes is determined. The first on the list are the candidates who received the most votes. By default, the parameter takes a null value.If `limit` is null, then this action returns total number of votes of all candidates from the `priority` list.

---

get_top

The `get_top` inline function returns a list of the most rated candidates, compiled according to a certain principle.

```
static inline std::vector<elected_t> get_top(
   symbol_code token_code,
   uint16_t elected_num,
   uint16_t reserve_num,
   bool strict = true
)
```

Parameters:

- `token_code` — symbol of tokens tokens by which the votes were defined.
- `elected_num` — the requested number of candidates who received the largest number of votes among those elected as validators.
- `reserve_num` — the requested number of reserve candidates who were not included in the list of eligible candidates. The list of reserve candidates includes those candidates who received the most votes among remaining candidates, taking into account the time when each of the candidates did not appear in the list of validators.
- `strict` — `true` ignores the votes of candidates who do not have a public key. Default is `true`.
  The `get_top` function returns a list of candidates made up of the requested number of `elected_num` elected to validators and the number of `reserve_num` backups added to it taking into account the strict parameter.