Publication

The list of actions implemented in the golos.publication smart contract

- contract setting: setlimit, setrules and setparams;
- message actions: createmssg, updatemssg, deletemssg, reblog, erasereblog, setcurprcnt and setmaxpayout;
- voting: upvote, downvote and unvote;
- system actions: closemssgs, calcrwrdwt, paymssgrwrd and deletevotes.

In addition, this contract contains logic for determining the payments to authors, curators and beneficiaries of posts.

Data types used

percentage
16-bit value with declaration types `int16_t` or `uint16_t` and accuracy to hundredths of a percent. The percentage is between -10000 and 10000 (for example: -10000 = -100 %, 0 = 0 %, 10000 = 100 %, 1234 = 12,34 %).
Negative percentage values are specified by the type `int16_t`.

mssgid
Message identifier, structure containing fields:

- `author` — message author;
- `permlink` — unique name of message within a particular author's publications.

```
struct mssgid {
name author;
std::string permlink;
}
```

beneficiary
Beneficiary, structure containing fields:

- `account` — beneficiary account;
- `weight` — *percentage*, a share of payment to a beneficiary. It is a part of an author reward amount.

```
struct beneficiary {
name account;
uint16_t weight;
};
```

---

setlimit

The `setlimit` action is used to set rules that restrict user operations. The mechanism for restricting user operations is based on interaction of two smart contracts — precisely `golos.publication` and `golos.charge`. Each action of the `golos.publication` smart contract is linked to a certain battery of the `golos.charge` smart contract. In the `setlimit` parameters, it needs to specify an action (for example, `createmssg` or `upvote`) and a battery to be linked to it.

The `setlimit` action has the following form:

```
setlimit(
    std::string act,
    symbol_code token_code,
    uint8_t     charge_id,
    int64_t     price,
    int64_t     cutoff,
    int64_t     vesting_price,
    int64_t     min_vesting
);
```

Parameters:

- `act` — a name of action. The contract supports the values `post` , `comment` , `vote` and `post bandwidth` .
- `token_code` — token code (character string that uniquely identifies a token).
- `charge_id` — battery ID. The action specified as `act` is limited to the charge of this battery. Multiple actions can be linked to a single battery. For voting actions such as `upvote` , `downvote` and `unvote` , the battery ID value should be set to zero.
- `price` — a price (in arbitrary units) of the consumed battery recourse with the `charge_id` identifier for the `act` action. The battery recourse is reduced after each performed action and recovered with time.
- `cutoff` — lower threshold value of the battery recourse at which the `act` action is blocked.
- `vesting_price` — amount of vesting, that the user must pay for performing the `act` action, in case of exhaustion of the battery recourse (reaching lower threshold value). The `act` action will be executed if the user allows to withdraw the specified amount of vesting from her/his balance. For payment it is necessary that on the user balance there was a necessary sum of vesting in unblocked state. *(**Note:** this parameter is currently disabled and should be equal to «0»).*
- `min_vesting` — minimum value of vesting that a user needs to have on her/his balance to perform the `act` action. *(**Note:** this parameter is currently disabled and should be equal to «0»).*

The interaction of smart publishing contracts and batteries allows a witness to flexibly configure restrictions on user actions (for example, such actions as voting for posts, publication of post and leaving comments can be correlated with the resources of three separate batteries. In this case, user activity will be limited for each of these actions. Also, all these actions can be linked to only one battery, that is, be limited by resources of the same battery). For each action performed by the user, she/he is charged a value corresponding to cost of the consumed battery recourse. When the `golos.charge` smart contract reaches the threshold value of used battery recourse, user's actions are blocked until necessary resource appears in the battery again.

---

setrules

The `setrules` action is used for setting rules that apply in application for distribution of rewards between authors and curators of posts.
The `setrules` action has the following form:

```
void setrules(
    funcparams  mainfunc,
    funcparams  curationfunc,
    funcparams  timepenalty,
    int64_t     maxtokenprop,
    symbol      tokensymbol
);
```

Parameters:

- `mainfunc` — a function that calculates a total amount of rewards for an author and post curators in accordance with accepted algorithm (for example, a linear algorithm or a square root algorithm). The algorithm used in the function is selected by witnesses voting. The function contains two parameters: mathematical expression (the algorithm itself) by which the reward is calculated, and maximum allowable value of argument for this function. When setting parameter values for `setrules`, they are checked for correctness (including for monotonous behavior and for non-negative value).
- `curationfunc` — a function that calculates a fee for each of the curators in accordance with accepted algorithm (similar to calculation for `mainfunc`).
- `timepenalty` — a function that calculates a weight of vote, taking into account the time of voting and the penalty time duration.
- `maxtokenprop` — the maximum share of the reward paid to an author in tokens (balance in vesting).
- `tokensymbol` — a token type (within the Golos application, only Golos tokens are used).

A transaction containing the `setrules` action must be signed by most leaders.

---

createmssg

The `createmssg` action is used to create a message as a response to a previously received (parent) message. The `createmssg` action has the following form:

```
void createmssg(
    mssgid        message_id,
    mssgid        parent_id,
    std::vector<structures::beneficiary> beneficiaries,
    int16_t       tokenprop,
    bool          vestpayment,
    std::string   headermssg,
    std::string   bodymssg,
    std::string   languagemssg,
    std::vector<structures::tag> tags,
    std::string   jsonmetadata,
    std::optional<uint16_t> curators_prcnt,
    std::optional<asset> max_payout
);
```

Parameters:

- `message_id` — identifier of the message.
- `parent_id` — identifier of the parent message. To create a post, the field `parent_id.author` should be empty.
- `beneficiaries` — list of beneficiaries. The list may be empty.
- `tokenprop` — a share of reward paid in liquid tokens (the balance is paid in vesting). This value cannot exceed the `maxtokenprop` value specified in `set_rules`.
- `vestpayment` — `true`, if a user gives permission to pay in vestings in case of battery resource exhaustion (the message is sent regardless of battery resource). Default value is `false`. *(**Note:** this parameter is currently disabled and should be equal to «false»).*
- `headermssg` — title of the message.
- `bodymssg` — body of the message.
- `languagemssg` — language of the message.
- `tags` — a list of tags.
- `jsonmetadata` — metadata in the JSON format.
- `curators_prcnt` — a share (in percent) of reward deducted to curators from total amount of rewards for the created message. The parameter value is set by the message author within the range of values set by leaders. This parameter is optional. If is not specified it takes default value — the value of min_curators_prcnt;
- `max_payout` — maximum possible reward amount for the message being paid out of the pool to which this message is linked. This amount is set by the author. It can be used to create messages without payouts. The parameter is optional and defaults to `asset::max_amount`.

The `parent_id` parameter identifies the parent message to which a response is created via `createmssg`.

To perform the `createmssg` action it is required that the transaction should be signed by the author of the message.

The key that is used to search for a message is bound to the `account` and `permlink` parameters.

---

updatemssg

The `updatemsg` action is used to update a message previously sent by user.
The `updatemssg` action has the following form:

```
void updatemssg(
    mssgid        message_id,
    std::string   headermssg,
    std::string   bodymssg,
    std::string   languagemssg,
    std::vector<structures::tag> tags,
    std::string   jsonmetadata
);
```

Parameters:

- `message_id` — identifier of the message being updated. The parameter contains the fields: `author` — author name of the message being updated, `permlink` — unique name of the message within publications of this author.
- `headermssg` — title of the message.
- `bodymssg` — body of the message.
- `languagemssg` — language of the message.
- `tags` — tag that is assigned to the message.
- `jsonmetadata` — metadata in the JSON format.

To perform the `updatemssg` action it is required that the transaction should be signed by the author of the message.

---

deletemssg

The `deletemssg` action is used to delete a message previously sent by user.
The `deletemssg` action has the following form:

```
void deletemssg(mssgid   message_id);
```

Parameter:

- `message_id` — identifier of the message to be deleted. The parameter contains the fields: `author` — author of the message to be deleted, `permlink` — unique name of the message within publications of this author.

The message cannot be deleted in the following cases:

- the message has a comment;
- total weight of all votes cast for this message is greater than zero. The weight of the user's vote depends on amount of vesting she/he uses. A message cannot be deleted if the total weight of all votes has a positive value.

To perform the `deletemssg` action it is required that the transaction should be signed by the author of the message.

---

upvote

The `upvote` action is used to cast a vote in the «upvote» form when voting for a message. The `upvote` action has the following form:

```
void upvote(
    name      voter,
    mssgid    message_id,
    uint16_t  weight
);
```

Parameters:

- `voter` — voting account name.
- `message_id` — identifier of the message.
- `weight` — the vote weight of the account name `voter`, *percentage*.

To perform the `upvote` action it is required that the transaction should be signed by the account name `voter` .

downvote

The `downvote` action is used to cast a vote in the «downvote» form when voting for a message. The `downvote` action has the following form:

```
void downvote(
    name       voter,
    mssgid     message_id,
    uint16_t   weight
);
```

Parameters:

- `voter` — voting account name.
- `message_id` — identifier of the message.
- `weight` — the vote weight of the account name `voter` , *percentage* (the weight value must be positive).

To perform the `downvote` action it is required that the transaction should be signed by the account name `voter` .

unvote

The `unvote` action is used to revoke user's own vote that was previously cast for the post. The `unvote` action has the following form:

```
void unvote(
    name     voter,
    mssgid   message_id
);
```

Parameters:

- `voter` — account name that revokes her/his own vote previously cast for the message.
- `message_id` — identifier of the message.

To perform the `unvote` action it is required that the transaction should be signed by the account name `voter` .

---

closemssgs

The `closemssgs` is a system action and used to «close» messages that have reached the time of payment.
The `closemssgs` action has the following form:

```
void close_messages(name payer)
```

Parameter:

- `payer` — account name paying for data storage.

---

reblog

The `reblog` action is used to place a post adopted from another author under this smart contract, as well as to add rebloger's own title and text to the post as a note.

The `reblog` action has the following form:

```
void reblog(
    name rebloger,
    mssgid message_id,
    std::string headermssg,
    std::string bodymssg
)
```

Parameters:

- `rebloger` — account name of the reblogger.
- `message_id` — identifier of the post-original.
- `headermssg` — title of the note to be added. This field can be empty.
- `bodymssg` — body of the note to be added. This field can be empty if the field `headermssg` is empty too.

Restrictions that are imposed on the `reblog` action:

- It is not allowed to perform `reblog` of own post, that is, the author of which is the account `rebloger`.
- If a title of the note `headermssg` is specified, then its body `bodymssg` must be present too (reblogging without a header, as well as without a header and body is allowed).
- The title length of the added note should not exceed 256 characters.

To perform the `reblog` action it is required that the transaction should be signed by the account name `rebloger`.

---

erasereblog

The `erasereblog` action is used to remove a previously posted reblog. The `erasereblog` action has the following form:

```
void erasereblog(
    name rebloger,
    mssgid message_id
)
```

Parameters:

- `rebloger` — account name removing the reblog.
- `message_id` — identifier of the post-original.

A transaction containing the `erasereblog` action should be signed by the `rebloger` account.

setcurprcnt

The `setcurprcnt` action is used by author of a post to set or change previously specified amount (in percent) of reward, allocated to the curators.

The `setcurprcnt` action has the following form:

```
void set_curators_prcnt(
    mssgid message_id,
    uint16_t curators_prcnt
)
```

Parameters:

- `message_id` — identifier of the post.
- `curators_prcnt` — a share (*percentage*) of reward deducted to curators from total amount of rewards for the post.

After the start of voting for a post, any change in the share of payment to curators is unacceptable.

To perform the `setcurprcnt` action it is required that the transaction should be signed by the post author `message_id.author` .

setmaxpayout

The `setmaxpayout` action is used by author of a message to set or change maximum possible payment to curators for the message.

The `setmaxpayout` action has the following form:

```
void setmaxpayout(
    mssgid message_id,
    asset max_payout
);
```

Parameters:

- `message_id` — identifier of the message for which amount the payment to curators is setting. The parameter contains the fields: `author` — author of the message, `permlink` — unique name of the message within publications of this author.
- `max_payout` — maximum possible reward amount for the message being paid out of the pool to which this message is linked. This amount is set by the author in the form of funds (tokens) that are in this pool.

The following restrictions apply to changing the `max_payout` parameter:

- the parameter can only be changed for open messages.
- the parameter can only be changed for messages that do not have votes.
- the parameter can only be decreased in relation to its previous value. It must be positive. Retaining old value of the parameter is unacceptable.

To perform the `setmaxpayout` action it is required that the transaction should be signed by the author of message.

---

calcrwrdwt

The `calcrwrdwt` action is internal and unavailable to the user. It is used to calculate a post weight based on number of publications made by its author for a certain time.

The action has the following form:

```
void calcrwrdwt(
    name account,
    int64_t mssg_id,
    int64_t post_charge
)
```

Parameters:

- `account` — account name that is the post author.
- `mssg_id` — internal identifier of the post.

- `post_charge` — current battery life. It is used to limit user activity in posting. Battery charge decreases with increasing number of posting by the author for a certain time. The amount paid for posts is also reduced.

To perform the `calcrwrdwt` action it is required that the transaction should be signed by the `golos.publication` smart contract account.

---

paymssgrwrd

The `paymssgrwrd` action is internal and unavailable to the user. It is used to pay rewards for a «closed» message to curators, beneficiaries and author. The action has the following form:

```
void paymssgrwrd(mssgid message_id)
```

Parameter:

- `message_id` — identifier of the message for which awards are paid.

Reward may be partially paid when calling `paymssgrwrd`. In this case, repeated calls of `paymssgrwrd` are allowed.

This action can be performed by any account.

---

deletevotes

The `deletevotes` is an inner action. It is used to free up memory occupied by voting records for messages for which rewards have been paid. The action has the following form:

```
void deletevotes(int64_t message_id, name author)
```

Parameters:

- `message_id` — internal message identifier.
- `author` — post author.

When calling `deletevotes`, the votes may be removed only partially not to completely. In this case, repeated calls of `deletevotes` are allowed.

A transaction containing the `deletevotes` action must be signed by the `golos.publication` contract account.

---

setparams

The `setparams` action is used to configure the parameters of `golos.publication` smart contract. The action has the following form:

```
void set_params(std::vector<posting_params> params)
```

Parameter:

- `params` — list of structures with variable contract parameters.

Other parameters which are used and set in the golos.publication smart contract

There are other parameters in the `golos.publication` smart contract that can be set by calling `set_params`:

- `cashout_window` — payout window options:
    - `window` — time interval after which a message reward is paid;
    - `upvote_lockout` — the time interval before «closing» a post. During this interval the positive voice «upvote» is prohibited, but allowed the negative voice «downvote».
- `max_beneficiaries` — maximum possible number of beneficiaries.
- `max_comment_depth` — maximum allowable nesting level of comments (it shows the allowed nesting level of child comments relative to root one).
- `social_acc` — account name of the `social` contact.

- `referral_acc` — account name of the `referral` contact.
- `curators_prcnt` — a share (in percent) of reward deducted to curators from total amount of rewards for a message. The parameter sets thresholds (`min_curators_prcnt` and `max_curators_prcnt`) within which the author can specify her/his own percentage value of curators fee.
- `bwprovider` — account of the `bandwidth` resource provider as well as the permission used (type `permission_level`).
- `min_abs_rshares` — minimum absolute value of `rshares` for voting operations. Votes with a lower value are rejected.