Social

Purpose of golos.social smart contract

The `golos.social` smart contract provides users the following features:

- creating and editing user profiles (metadata).
- pin-list establishment that allows its owner to receive information about the publications of users that he/she is interested in.
- the establishment of a so-called «black» list allowing to block communication between the owner of this list and any unwanted users.

Terminology used in the description of golos.social smart contract

User profile — user metadata stored in the client application database in the form of a structure characterizing the user. The user profile is created and edited by the user. The `golos.social` smart contract is not responsible for the storing of user metadata, but only controls whether the user has the right to change or delete metadata.

A user's pin list — a database item containing a list of account names that a given user is interested in. The user's pin list is created and edited by the user. It can be used in the client application to create subscriptions, including informing the given user (subscriber) about the appearance of a new post whose author's name is contained in the pin list.

«Black» list — a database item containing a list of account names that this user characterizes as unwanted. The «black» list is created and edited by the user. Using the «black» list allows the user to block comments and votes of accounts whose names are contained in this list. The `golos.publication` smart contract verifies the absence of the account name in this list when performing `createmssg` action.

Actions supported by golos.social smart contract

The `golos.social` smart contract supports the following actions: pin, unpin, block, unblock, updatemeta and deletemeta.

---

accountmeta type

The `accountmeta` type is intented to describe the user profile and comes in the following structure:

```
struct accountmeta {
  optional<std::string> type;                //  Type

    optional<std::string> app;               // Mother application

    optional<std::string> email;             // User's e-mail address
    optional<std::string> phone;             // User's phone number
    optional<std::string> facebook;          // User's facebook account
    optional<std::string> instagram;         // User's instagram account
    optional<std::string> telegram;          // User's telegram account
    optional<std::string> vk;                // User's vk account
    optional<std::string> whatsapp;          // User's whatsapp account
    optional<std::string> wechat;            // User's wechat account
    optional<std::string> website;           // Personal website url

    optional<std::string> first_name;        // User first name
    optional<std::string> last_name;         // User surname
    optional<std::string> name;              // Account name
    optional<std::string> birth_date;        // Date of birth
    optional<std::string> gender;            // Gender
    optional<std::string> location;          // Province of residence
    optional<std::string> city;              // City of residence

    optional<std::string> about;             // About a user
    optional<std::string> occupation;        // Occupation
    optional<std::string> i_can;             // User capability
    optional<std::string> looking_for;       // User purpose
    optional<std::string> business_category; // Business category

    optional<std::string> background_image; // Background image
    optional<std::string> cover_image;       // Cover image
    optional<std::string> profile_image;     // Profile (avatar) picture
    optional<std::string> user_image;        // User image
    optional<std::string> ico_address;       // Ico-address

    optional<std::string> target_date;       // Accomplish due date
```

```
    optional<std::string> target_plan;      // User target
    optional<std::string> target_point_a;   // Intermediate target A
    optional<std::string> target_point_b;   // Intermediate target B
}
```

The `accountmeta` type is used as a parameter in the `updatemeta` action.

---

pinblock table

The `pinblock` table is a database item and contains information about the relationship of one user to another. The data from the table is used to create a pin-list and a list of blocked users.

The `pinblock` table contains the following fields:

- `(name) SERVICE.scope` — the account name (the first name taken from a pair), which puts a user name of interest into one of the lists — the pin-list or the list of blocked users;
- `(name) account` — the account name that was added to one of the lists — the pin-list or the list of blocked accounts;
- `(bool) pinning` — «true» means that the account name has been added to the pin list;
- `(bool) blocking` — «true» means that the account name has been added to the list of blocked accounts.

> **Please note:**
> Both `pinning` and `blocking` fields cannot be set to «true».
> The record is removing from the table if the `pinning` and `blocking` fields are «false».

---

pin

The `pin` action is used to add an account name of interest to a pin-list. This action can only be performed by the pin-list owner. The `pin` action has the following form:

```
void pin(
    name pinner,
    name pinning
);
```

Parameters:

- `pinner` — account name which is the pin-list owner and adds the name specified in the `pinning` field to the pin-list.
- `pinning` — the account name that is being added to the pin-list.

The rights to run the `pin` action belong to `pinner` account.

The `pin` action is introducing the following restrictions:

- a user is not allowed to add own name to the pin-list, that is, the `pinning` field must not contain the `pinner` field value.
- it is not allowed to add account name to the pin-list if pin-list already contains this name.
- it is not allowed to add account name to the pin-list if this name has been blocked by the `pinner` account, that is, this name is in the list of blocked accounts.

---

unpin

The `unpin` action is used to remove an account name from a pin-list. The `unpin` action is as follows:

```
void unpin(
    name pinner,
    name pinning
);
```

Parameters:

- `pinner` — the account name that removes a name specified in the `pinning` field from the pin-list.
- `pinning` — the account name that is about to be removed from the pin-list.

The rights to run the `pin` action belong to `pinner` account.

The `unpin` action is introducing the following restrictions:

- it is not allowed to remove the account name which is the pin-list owner, that is, the `pinning` field must never be equal to `pinner` filed.
- it is not allowed to remove an account name that is not present in the pin-list.

---

block

The `block` action is used to add an account name of interest to «black» list. This action can only be performed by the «black» list owner.
The `block` action has the following form:

```
void block(
    name blocker,
    name blocking
);
```

Parameters:

- `blocker` — account name which is the «black» list owner and adds the name specified in the `blocking` field to the «black» list.
- `blocking` — the account name that is being added to the «black» list.

The rights to run the `block` action belong to `blocker` account.

The `block` action is introducing the following restrictions:

- a user is not allowed to block him/herself, that is, the `blocking` field must not be equal to `blocker` field value.
- it is not allowed to add account name to the black-list if black-list already contains this name.

---

unblock

The `unblock` action is used to remove an account name from a «black» list.
The `unblock` action has the following form:

```
void unblock(
    name blocker,
    name blocking
);
```

Parameters:

- `blocker` — the account name that removes a name specified in the `blocking` field, from the «black» list.
- `blocking` — the account name that is about to be removed from the «black» list.

The rights to run the `unblock` action belong to `blocker` account.

The `unblock` action is introducing the following restrictions:

- it is not allowed to remove the account name which is the «black» list owner, that is, the `blocking` field must never be equal to `blocker` field;
- it is not allowed to remove an account name that is not present in the «black» list.

---

updatemeta

The `updatemeta` action is used to fill, update, or delete account profile field values.
The `updatemeta` action has the following form:

```
void updatemeta(
    name account,
    accountmeta meta
);
```

Parameters:

- `account` — name of the account whose profile is being edited.

- `meta` — a value of type of the `accountmeta` structure.

The rights to run the `updatemeta` action belong to the account that is specified in the `account` field. The updatemeta action does not interact with the database. It only checks the rights of the user who changes her/his profile. Updating user metadata within the database should be implemented in the client application.

---

deletemeta

The `deletemeta` action is used for deleting an account profile.
The `delelemeta` action has the following form:

```
void deletemeta(name account);
```

The parameter `account` is a name of the account whose profile is being deleted.

The rights to run `deletemeta` action belong to the account that is deleting her/his profille. The `deletemeta` action does not interact with the database. It only checks the rights of the user who is about to delete her/his profile.
Removal of user metadata from the database should be implemented in the client application.