BIOS

Purpose of the cyber.bios smart contract

The `cyber.bios` smart contract is used as a link between operations, executed both directly in smart contracts and the node core.

`cyber.bios` includes the following actions: newaccount, setprods, setparams, reqauth, setabi, setcode, onblock, checkwin, bidname, bidrefund, canceldelay, updateauth, deleteauth, linkauth and unlinkauth.

newaccount

The `newaccount` action is used when creating new accounts in the system.

```
void bios::newaccount(
    name creator,
    name name,
    authority owner,
    authority active
)
```

Parameters:

- `creator` — new account creator.
- `name` — the account name that is created in the system.
- `owner` — structure of type `authority` containing `owner public key` for the new account.
- `active` — structure of type `authority` containing `active public key` for the new account.

Restrictions:

- The name created (not redeemed at auction) must contain no more than 12 characters. Also, it must not contain the dot symbol.
- The name purchased at auction or created through the system account can be expanded by adding characters to the right after the dot symbol (for example, an owner of the name `cyber` can expanded it to the name `cyber.anyname`).

A transaction containing `newaccount` action must be signed by the new account creator.

---

setprods

The action `setprods` is used when creating a block production schedule. This schedule contains a list of validators in accordance with their rating. This action is not accessible to a user and is called by the system. This action takes the following form:

```
[[eosio::action]] void setprods(eosio::producer_key schedule )
```

The `schedule` parameter is a list of validator keys.

The `cyber` account signature is required to execute a transaction containing the `setprods` action (`cyber` is account of `cyber.bios` smart contract).

---

setparams

The `setparams` action is used to configure system parameters. This action is not accessible to a user and is called by the system. This action takes the following form:

```
[[eosio::action]] void setparams( eosio::blockchain_parameters params )
```

The `params` parameter is a structure containing settable system parameters.

The `cyber` account signature is required to execute a transaction containing the `setparams` action.

---

reqauth

The `reqauth` action is used to verify the user's signature in a transaction. This action takes the following form:

```
[[eosio::action]] void reqauth( name from )
```

The `from` parameter is the account whose signature is verified.

The `from` account signature is required to execute a transaction containing the `reqauth` operation.

---

setabi

The `setabi` action is used to upload ABI-description to an account. This action takes the following form:

```
[[eosio::action]] void setabi(
    name account,
    std::vector<char> abi
)
```

Parameters:

- `account` — the account to which ABI-description will be downloaded.
- `abi` — an array of bytes containing ABI-description.

---

setcode

The `setcode` action is used to upload smart contract code to an account. The signature of this action is:

```
[[eosio::action]] void setcode(
    name account,
    uint8_t vmtype,
    uint8_t vmversion,
    std::vector<char> code
)
```

Parameters:

- `account` — the account to which smart contract code will be downloaded.
- `vmtype` — type of contract (type of virtual machine). In this release, the parameter is set to «0».
- `vmversion` — version of contract (version of virtual machine). In this release, the parameter is set to «0».
- `code` — an array of bytes containing smart contract code.

---

onblock

The `onblock` action is inaccessible to a user and is called by the system each time after creation of a new block. Inside `onblock`, the `cyber.govern` smart contract code is called. This action takes the following form:

```
void bios::onblock(block_header header)
```

The `header` parameter is a block header.

---

checkwin

Action `checkwin` is used to register a name owner (a winner) at auction. This action has no parameters and is called implicitly.

```
void bios::checkwin()
```

---

bidname

The bidname action allows a user to bid on a specific name at the names auction. This action takes the following form:

```
void bios::bidname(
    name bidder,
    name newname,
    eosio::asset bid
)
```

Parameters:

- `bidder` — an account which bids at the action.
- `newname` — betting name (a name on which the bid is done).
- `bid` — a bet on the name (a structure value specified the bid). Bets are accepted only in system tokens (CYBER).

The `bidder` account will be announced as an owner (a winner) of `newname`, if:

- the `bid` is the highest in the name newname;
- the `bid` remains the highest after 24 hours.
  In this case, the bid is not returned to the bidder. If the `bid` is beaten within 24 hours, it will be returned back to the `bidder` (in this release, the bid is not automatically refunded; a call of `bidrefund` is required to refund it).

---

bidrefund

The `bidrefund` action is used to refund a non-winning (not the highest) bid to an auction participant

```
void bios::bidrefund( name bidder )
```

The `bidder` parameter is an auction participant to whom the bid is returned.

The bid is automatically refunded to `bidder` if the `bidder` does not become a winner. The bidder should create a transaction with the bidrefund operation to refund the bid. One calling `bidrefund` is enough to return all of non-winning bids in case the bidder put several bids on different names and neither of these bids (or part of them) were non-winning.

---

canceldelay

The `canceldelay` action cancels a deferred transaction.

```
void canceldelay(
    permission_level canceling_auth,
    eosio::checksum256 trx_id
)
```

Parameters:

- `canceling_auth` — value containing an account of the transaction creator and its permission (permission name takes the value `active`).
- `trx_id` — a deferred transaction identifier.

The `canceling_auth` account signature is required to execute a transaction containing the `canceldelay` operation.

---

updateauth

The `updateauth` action is used to add or change authorization for an account. This action takes the following form:

```
void updateauth(
    name  account,
    name  permission,
    name  parent,
    authority auth
)
```

Parameters:

- `account` — an account name whose authorization changes.
- `permission` — a permission name (for example: owner, active, posting, etc.) to be changed.
- `parent` — a name of parent permission (for example: `owner` is the parent permission to `active`; `active` is the parent permission to `posting`. Child permissions cannot exceed parental permissions). In most cases, this parameter is set to `active'.
- `auth` — structure of the form `authority`, the new values of which should match the `permission`.

The `account` signature is required to execute a transaction containing the `updateauth` operation.

---

deleteauth

The action `deleteauth` is used to delete authorization of an account. This action takes the following form:

```
void deleteauth(
    name account,
    name permission
)
```

Parameters:

- `account` — account name whose authorization is being deleted.
- `permission` — permission name that is deleted.

The `account` signature is required to execute a transaction containing the `deleteauth` operation.

---

linkauth

The `linkauth` action allows the `permission` to execute some action in a specific contract.

```
void linkauth(
    name account,
    name code,
    name type,
    name requirement
)
```

Parameters:

- `account` — account name for which permission is changed.
- `code` — an account name of the contract in which some action will be executed.
- `type` — type of permission. Any action can be performed in the contract if this parameter is empty, otherwise only one operation specified in the parameter can be performed (parameter value matches the name of the action).
- `requirement` — name of `permission`, which is allowed to perform the operation. This parameter cannot be empty.

The parameters `account` and `requirement` specify `permission_level`.
The parameters `code` and `type` specify the contract account and action, respectively.

> **Notes:**
> Any set of `account`, `code`, and `type` parameters must correspond to only one `requirement` value. This means that one account cannot have two or more different permissions related to the same contract's account and action.

> If the same `account`, `code` and `type` values are passed to the action again, but with a different `requirement`, its previous value will be replaced by the last passed.

The `account` signature is required to execute a transaction containing the `linkauth` operation.

---

unlinkauth

The `unlinkauth` action removes the name of `permission` provided by the `linkauth` action in a specific contract

```
void unlinkauth(
    name account,
    name code,
    name type
)
```

Parameters:

- `account` — a name of the account whose permission is removed.
- `code` — an account name of the contract in which some action was allowed to execute.
- `type` — type of permission..

The `account` signature is required to execute the transaction containing the `unlinkauth` operation.