

- [AdminControl](#)
- [SponsorWhitelistControl](#)
- [Staking](#)
- [ConfluxContext](#)
- [PoSRegister](#)
- [CrossSpaceCall](#)
- [ParamsControl](#)

AdminController

AdminController

admin

admin setAdmin(address contractAddr, address newAdmin)

destroy(address contractAddr) suicide() SponsorWhitelist

ConfluxScan

AdminController getAdmin(address contractAddr)

1. A B C C
2. A B B C D C A C B
3. Conflux D 2

contract_addr AdminControl.setAdmin(contract_addr, new_admin)

AdminControl.destroy(contract_addr)

```
const PRIVATE_KEY = '0xxxxxxx';
const cfx = new Conflux({
  url: 'http://test.confluxrpc.org',
  logger: console,
});
const account = cfx.wallet.addPrivateKey(PRIVATE_KEY); // create account instance

const admin_contract = cfx.InternalContract('AdminControl')
// to change administrator
admin_contract.setAdmin(contract_addr, new_admin).sendTransaction({
```

```
    from: account,  
  }).confirmed();  
  
// to kill the contract  
admin_contract.destroy(contract_addr).sendTransaction({  
  from: account,  
}).confirmed();
```

SponsorWhitelistControl

Conflux

Dapps

SponsorWhitelistControl

Message Call

A

C

B

fl

B

C

, Conflux

A

B

A

B

B

C

B

 /

SponsorControl

•

sponsor_for_gas

•

sponsor_for_collateral

•

sponsor_balance_for_gas

•

sponsor_balance_for_collateral

•

sponsor_limit_for_gas_fee

•

whitelist

• *:

sponsor_for_gas

whitelist

sponsor_limit_for_gas_fee

sponsor_balance_for_gas

sponsor_balance_for_gas

• :

sponsor_balance_for_collateral

whitelist

sponsor_balance_for_collateral

sponsor_for_gas

sponsor_for_collateral

sponsor_limit_for_gas_fee

SponsorControl

sponsor_for_gas

setSponsorForGas(address contractAddr, uint upperBound)

1.

sponsor_balance_for_gas

2.

sponsor_limit_for_gas_fee

upperBound

sponsor_balance_for_gas

3.

1000

1000

sponsor_balance_for_gas

sponsor_for_gas

sponsor_balance_for_gas

sponsor_for_gas

sponsor_limit_for_gas_fee

sponsor_for_collateral

setSponsorForCollateral(address contractAddr)

sponsor_for_collateral

sponsor_balance_for_collateral

```
setSponsorForGas(address contractAddr, uint upperBound) setSponsorForCollateral( address
contractAddr) sponsor_limit_for_gas_fee
```

```
addPrivilege(address[] memory) sponsor_for_gas 0x0000000000000000000000000000000000000000000000000000000000000000
removePrivilege(address[] memory)
```

1.

```
addPrivilegeByAdmin(address contractAddr, address[] memory addresses)
removePrivilegeByAdmin(address contractAddr, address[] memory addresses)
```

```
pragma solidity >=0.4.15;

import "https://github.com/Conflux-Chain/conflux-
rust/blob/master/internal_contract/contracts/SponsorWhitelistControl.sol";

contract CommissionPrivilegeTest {
    mapping(uint => uint) public ss;

    function add(address account) public payable {
        SponsorWhitelistControl cpc =
SponsorWhitelistControl(0x0888000000000000000000000000000000000000000000000000000000000001);
        address[] memory a = new address[](1);
        a[0] = account;
```



```
you_contract.add(white_list_addr).sendTransaction({  
  from: account,  
})  
  
you_contract.remove(white_list_addr).sendTransaction({  
  from: account,  
})
```

whitelist

you_contract.foo()

you_contract.par_add(1, 10)

Staking

Conflux

Staking

deposit(uint amount) amount balance stakingBalance . payable

withdraw(uint amount)

Conflux

voteLock(uint amount, uint unlock_block_number) stakingBalance unlock_block_number amount

stakingBalance X 5CFX **

x y Ccy 2 * 60 * 60 * 24 * 365 x

1. stakingBalance 10CF voteLock(100 * 10^18, x) 1stakingBalance

2. voteLock(8 * 10^18, x)

3. voteLock(6 * 10^18, x+y) 2CFX 6CFX

4. voteLock(0, x) 2 3

5. voteLock(9 * 10^18, x+y) " 9CFX "

[Conflux Protocol Specification](#) 8.3.2

```
const PRIVATE_KEY = '0xxxxxxx';
const cfx = new Conflux({
  url: 'http://test.confluxrpc.org',
  logger: console,
```



```
});  
  
const account = cfx.wallet.addPrivateKey(PRIVATE_KEY); // create account instance  
  
const staking_contract = cfx.InternalContract(' Staking' );  
// deposit some amount of tokens  
staking_contract.deposit(your_number_of_tokens).sendTransaction({  
  from: account,  
}).confirmed();  
  
// withdraw some amount of tokens  
staking_contract.withdraw(your_number_of_tokens).sendTransaction({  
  from: account,  
}).confirmed();  
  
// lock some tokens until some block number  
staking_contract.voteLock(your_number_of_tokens, your_unlock_block_number).sendTransaction({  
  from: account,  
}).confirmed();
```

ConfluxContext

ConfluxContext

epochNumber, posHeight, finalizedEpochNumber

```
pragma solidity >=0.4.15;

contract ConfluxContext {
    /** Query Functions */
    /**
     * @dev get the current epoch number
     * @return the current epoch number
     */
    function epochNumber() public view returns (uint256) {}
    /**
     * @dev get the height of the referred PoS block in the last epoch
     * @return the current PoS block height
     */
    function posHeight() public view returns (uint256) {}
    /**
     * @dev get the epoch number of the finalized pivot block.
     * @return the finalized epoch number
     */
    function finalizedEpochNumber() public view returns (uint256) {}
}
```

PoSRegister

PoSRegister

PoW

PoS

PoW

PoS

1. conflux
- PoW
- PoS
- PoW
2. CFX 1000
3. 1 PoS = 1000 CFX
4. vote power
5. `voteLock()` ng

- `staking`
- 1000CFX

register

PoW

PoS

,

PoW

1000CFX

PoS

PoW

1000CFX

1 vote power

- PoS
- `conflux rpc local pos register --power 1, register`

- identifier: PoS
- votePower
- blsPubKey: PoS blsPubKey
- vrfPubKey PoS vrfPubKey
- blsPubKeyProof: PoS blsPubKeyProof

increaseStake

PoW PoS , , CFX

- PoW PoS
- CFX 1000CFX

- votePower:

retire

PoW PoS , PoW PoS

- PoW PoS

- votePower:

getVotes

token votes token votes

- PoW PoS

- identifier: PoS

- totalStakedVotes token votes
- totalUnlockedVotes token votes

identifierToAddress

PoS PoW

- PoW PoS
- identifier: PoS
- address PoW

addressToIdentifier

PoW PoS

- PoW PoS
- address: PoW

- identifier PoS

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0;

interface PoSRegister {
    /**
     * @dev Register PoS account
     * @param indentifier - PoS account address to register
     * @param votePower - votes count
     * @param blsPubKey - BLS public key
     * @param vrfPubKey - VRF public key
     * @param blsPubKeyProof - BLS public key's proof of legality, used to against some
    attack, generated by confluent-rust fullnode
     */
    function register(
        bytes32 indentifier,
        uint64 votePower,
        bytes calldata blsPubKey,
        bytes calldata vrfPubKey,
        bytes[2] calldata blsPubKeyProof
    ) external;

    /**
     * @dev Increase specified number votes for msg.sender
     * @param votePower - count of votes to increase
     */
    function increaseStake(uint64 votePower) external;

    /**
     * @dev Retire specified number votes for msg.sender
     * @param votePower - count of votes to retire
     */
}
```

```

function retire(uint64 votePower) external;

/**
 * @dev Query PoS account's lock info. Include "totalStakedVotes" and "totalUnlockedVotes"
 * @param identifier - PoS address
 */
function getVotes(bytes32 identifier) external view returns (uint256, uint256);

/**
 * @dev Query the PoW address binding with specified PoS address
 * @param identifier - PoS address
 */
function identifierToAddress(bytes32 identifier) external view returns (address);

/**
 * @dev Query the PoS address binding with specified PoW address
 * @param addr - PoW address
 */
function addressToIdentifier(address addr) external view returns (bytes32);

/**
 * @dev Emitted when register method executed successfully
 */
event Register(bytes32 indexed identifier, bytes blsPubKey, bytes vrfPubKey);

/**
 * @dev Emitted when increaseStake method executed successfully
 */
event IncreaseStake(bytes32 indexed identifier, uint64 votePower);

/**
 * @dev Emitted when retire method executed successfully
 */
event Retire(bytes32 indexed identifier, uint64 votePower);
}

```


CrossSpaceCall

CrossSpaceCall conflux coreSpace ESpace coreSpace ESpace

- 1. conflux CoreSpace ESpace conflux
- 2. CoreSpace ESpace

- Conflux

createEVM

espace call

call createEVM nce+1 call

- init

-

transferEVM

PoW Espace

- conflux

- to: PoW Espace

-

callEVM

ESpace

call

- to: PoW Espace

- data: encode

•

staticCallEVM

ESpace

- to: CroeSpace ESpace
- data: encode

•

withdrawFromMapped

ESpace

CoreSpace

- value:

mappedBalance

CoreSpace CoreSpace

- address: CoreSpace

-

mappedNonce

CoreSpace CoreSpace nonce

- address: CoreSpace

- nonce

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0;
```

```
interface CrossSpaceCall {

    event Call(bytes20 indexed sender, bytes20 indexed receiver, uint256 value, uint256 nonce,
bytes data);

    event Create(bytes20 indexed sender, bytes20 indexed contract_address, uint256 value,
uint256 nonce, bytes init);

    event Withdraw(bytes20 indexed sender, address indexed receiver, uint256 value, uint256
nonce);

    event Outcome(bool success);

    function createEVM(bytes calldata init) external payable returns (bytes20);

    function transferEVM(bytes20 to) external payable returns (bytes memory output);

    function callEVM(bytes20 to, bytes calldata data) external payable returns (bytes memory
output);

    function staticCallEVM(bytes20 to, bytes calldata data) external view returns (bytes
memory output);

    function withdrawFromMapped(uint256 value) external;

    function mappedBalance(address addr) external view returns (uint256);

    function mappedNonce(address addr) external view returns (uint256);
}
```

ParamsControl

ParamsControl

DAO
Conflux

ParamsControl

PoW
CPH94

PoS

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.8.0;

interface ParamsControl {
    struct Vote {
        uint16 topic_index;
        uint256[3] votes;
    }

    /**
     * @dev cast vote for parameters
     * @param vote_round The round to vote for
     * @param vote_data The list of votes to cast
     */
    function castVote(uint64 vote_round, Vote[] calldata vote_data) external;

    /**
     * @dev read the vote data of an account
     * @param addr The address of the account to read
     */
    function readVote(address addr) external view returns (Vote[] memory);

    /**
     * @dev Current vote round
     */
    function currentRound() external view returns (uint64);

    /**
```

```
* @dev read the total votes of given round
* @param vote_round The vote number
*/
function totalVotes(uint64 vote_round) external view returns (Vote[] memory);

/**
* @dev read the PoS stake for the round.
*/
function posStakeForVotes(uint64) external view returns (uint256);

event CastVote(uint64 indexed vote_round, address indexed addr, uint16 indexed
topic_index, uint256[3] votes);
event RevokeVote(uint64 indexed vote_round, address indexed addr, uint16 indexed
topic_index, uint256[3] votes);
}
```