AElf

Release 0.6.0

Jun 19, 2020

Introduction

CHAPTER 1

todo

CHAPTER 2

Smart Contract APIs

This section gives an overview of some important contracts and contract methods. It's not meant to be exhaustive. With every method description we give the parameter message in JSON format, this can be useful when using client (like **aelf-command**).

2.1 Association Contract

2.1.1 CreateOrganization

```
rpc CreateOrganization (CreateOrganizationInput) returns (aelf.Address) { }

message CreateOrganizationInput {
    OrganizationMemberList organization_member_list = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}
message OrganizationMemberList {
    repeated aelf.Address organization_members = 1;
}

message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates an organization and returns its address.

- **CreateOrganizationInput**
  - **organizer member list**:
    - **organization_members**: initial organization members.

- **ProposalReleaseThreshold**:
  - **minimal approval threshold**: the value for the minimum approval threshold.
  - **maximal rejection threshold**: the value for the maximal rejection threshold.
  - **maximal abstention threshold**: the value for the maximal abstention threshold.
  - **minimal vote threshold**: the value for the minimal vote threshold.
- **ProposerWhiteList**:
  - **proposers**: proposer whitelist.

Returns

- **Address**: the address of newly created organization.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization

2.1.2 CreateOrganizationBySystemContract

```
rpc CreateOrganizationBySystemContract(CreateOrganizationBySystemContractInput) returns␣
↪(aelf.Address){}

message CreateOrganizationBySystemContractInput {
    CreateOrganizationInput organization_creation_input = 1;
    string organization_address_feedback_method = 2;
}

message CreateOrganizationInput{
    OrganizationMemberList organization_member_list = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}

message OrganizationMemberList {
    repeated aelf.Address organization_members = 1;
}

message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates an organization by system contract and returns its address.

- **CreateOrganizationBySystemContractInput**
  - **CreateOrganizationInput**
    - **OrganizationMemberList**
      - **organization_members**: initial organization members.

    - **ProposalReleaseThreshold**
        - **minimal approval threshold**: the value for the minimum approval threshold.
        - **maximal rejection threshold**: the value for the maximal rejection threshold.
        - **maximal abstention threshold**: the value for the maximal abstention threshold.
        - **minimal vote threshold**: the value for the minimal vote threshold.
    - **ProposerWhiteList**:
        - **proposers**: proposer whitelist.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

Returns

- **Address**: the address of newly created organization.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization.

2.1.3 ChangeOrganizationMember

```
rpc ChangeOrganizationMember(OrganizationMemberList) returns (google.protobuf.Empty) { }

message OrganizationMemberList {
    repeated aelf.Address organization_members = 1;
}

message OrganizationMemberChanged {
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    OrganizationMemberList organization_member_list = 2;
}
```

Changes the members of the organization. Note that this will override the current list.

- **OrganizationMemberList**:
  - **organization_members**: the new members.

After a successful execution, an **OrganizationMemberChanged** event log can be found in the transaction result.

OrganizationMemberChanged:

- **organization_address**: the organization address.
- **organization_member_list**: the new member list.

2.1.4 ACS3 specific methods

CreateProposal

```
rpc CreateProposal (CreateProposalInput) returns (aelf.Hash) { }

message CreateProposalInput {
    string contract_method_name = 1;
    aelf.Address to_address = 2;
    bytes params = 3;
    google.protobuf.Timestamp expired_time = 4;
    aelf.Address organization_address = 5;
    string proposal_description_url = 6,
    aelf.Hash token = 7;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

This method creates a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract.

- **CreateProposalInput**
  - **contract method name**: the name of the method to call after release.
  - **to address**: the address of the contract to call after release.
  - **expiration**: the timestamp at which this proposal will expire.
  - **organization address**: the address of the organization.
  - **proposal_description_url**: the url is used for proposal describing.
  - **token**: the token is for proposal id generation and with this token, proposal id can be calculated before proposing.

Returns

- **Hash**: id of the newly created proposal.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

Events

- **ProposalCreated**
  - **proposal_id**: id of the created proposal.

Approve

```
    rpc Approve (aelf.Hash) returns (google.protobuf.Empty) {}

    message ReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
```

(continues on next page)

(continued from previous page)

```
        aelf.Address address = 2;
        string receipt_type = 3;
        google.protobuf.Timestamp time = 4;
    }
```

This method is called to approve the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address.
  - **receipt type**: Approve.
  - **time**: timestamp of this method call.

Reject

```
    rpc Reject(aelf.Hash) returns (google.protobuf.Empty) { }

    message ReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
        aelf.Address address = 2;
        string receipt_type = 3;
        google.protobuf.Timestamp time = 4;
    }
```

This method is called to reject the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address.
  - **receipt type**: Reject.
  - **time**: timestamp of this method call.

Abstain

```
    rpc Abstain(aelf.Hash) returns (google.protobuf.Empty) { }

    message ReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
        aelf.Address address = 2;
        string receipt_type = 3;
        google.protobuf.Timestamp time = 4;
    }
```

This method is called to abstain from the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address.
  - **receipt type**: Abstain.
  - **time**: timestamp of this method call.

Release

```
    rpc Release(aelf.Hash) returns (google.protobuf.Empty) { }
```

This method is called to release the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address who votes for abstention.
  - **receipt type**: Abstain.
  - **time**: timestamp of this method call.

ChangeOrganizationThreshold

```
rpc ChangeOrganizationThreshold(ProposalReleaseThreshold) returns (google.protobuf.
↪Empty) { }

message ProposalReleaseThreshold {
    int64 minimal_approval_threshold = 1;
    int64 maximal_rejection_threshold = 2;
```

(continues on next page)

(continued from previous page)

```
    int64 maximal_abstention_threshold = 3;
    int64 minimal_vote_threshold = 4;
}

message OrganizationThresholdChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposalReleaseThreshold proposer_release_threshold = 2;
}
```

This method changes the thresholds associated with proposals. All fields will be overwritten by the input value and this will affect all current proposals of the organization. Note: only the organization can execute this through a proposal.

- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the new value for the minimum approval threshold.
  - **maximal rejection threshold**: the new value for the maximal rejection threshold.
  - **maximal abstention threshold**: the new value for the maximal abstention threshold.
  - **minimal vote threshold**: the new value for the minimal vote threshold.

After a successful execution, an **OrganizationThresholdChanged** event log can be found in the transaction result.

Events

- **OrganizationThresholdChanged**
  - **organization_address**: the organization address.
  - **proposer_release_threshold**: the new release threshold.

ChangeOrganizationProposerWhiteList

```
rpc ChangeOrganizationProposerWhiteList(ProposerWhiteList) returns (google.protobuf.
↪Empty) { }

message ProposerWhiteList {
    repeated aelf.Address proposers = 1;
}

message OrganizationWhiteListChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposerWhiteList proposer_white_list = 2;
}
```

This method overrides the list of whitelisted proposers.

- **ProposerWhiteList**:
  - **proposers**: the new value for the proposer whitelist.

After a successful execution, a **OrganizationWhiteListChanged** event log can be found in the transaction result.

Events

- **OrganizationWhiteListChanged**
  - **organization_address**: the organization address.
  - **proposer_white_list**: the new proposer whitelist.

CreateProposalBySystemContract

```
rpc CreateProposalBySystemContract(CreateProposalBySystemContractInput) returns (aelf.
↪Hash) { }

message CreateProposalBySystemContractInput {
    acs3.CreateProposalInput proposal_input = 1;
    aelf.Address origin_proposer = 2;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**
  - **CreateProposalInput**
    - **contract method name**: the name of the method to call after release.
    - **to address**: the address of the contract to call after release.
    - **expiration**: the date at which this proposal will expire.
    - **organization address**: the address of the organization.
    - **proposal_description_url**: the url is used for proposal describing.
    - **token**: the token is for proposal id generation and proposal id can be calculated before proposing.
  - **origin proposer**: the actor that trigger the call.

Returns

- **Address**: newly created organization address.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

Events

- **ProposalCreated**:
  - **proposal_id**: id of the created proposal.

ClearProposal

```
rpc ClearProposal(aelf.Hash) returns (google.protobuf.Empty) { }
```

Removes the specified proposal.

- **Hash**: id of the proposal to be cleared.

ValidateOrganizationExist

```
rpc ValidateOrganizationExist(aelf.Address) returns (google.protobuf.BoolValue) { }
```

Checks the existence of an organization.

- **Address**: organization address to be checked.

Returns

- **BoolValue**: indicates whether the organization exists.

2.1.5 View methods

GetOrganization

```
rpc GetOrganization (aelf.Address) returns (Organization) { }

message Organization {
    OrganizationMemberList organization_member_list = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
    aelf.Address organization_address = 4;
    aelf.Hash organization_hash = 5;
}
```

Returns the organization with the specified address.

- **Address**: organization address.

Returns

- **Organization**
  - **member list**: original members of this organization.
  - **ProposalReleaseThreshold**
    - **minimal approval threshold**: the value for the minimum approval threshold.
    - **maximal rejection threshold**: the value for the maximal rejection threshold.
    - **maximal abstention threshold**: the value for the maximal abstention threshold.
    - **minimal vote threshold**: the value for the minimal vote threshold.
  - **ProposerWhiteList**
    - **proposers**: proposer list.
  - **address**: the organizations address.
  - **hash**: the organizations id.

CalculateOrganizationAddress

```
rpc CalculateOrganizationAddress(CreateOrganizationInput) returns (aelf.Address){}

message CreateOrganizationInput {
    string token_symbol = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}
```

Calculates with input and returns the organization address.

- **CreateOrganizationInput**
  - **organizer member list**
    - **organization_members**: initial organization members.
  - **ProposalReleaseThreshold**:
    - **minimal approval threshold**: the value for the minimum approval threshold.
    - **maximal rejection threshold**: the value for the maximal rejection threshold.
    - **maximal abstention threshold**: the value for the maximal abstention threshold.
    - **minimal vote threshold**: the value for the minimal vote threshold.
  - **ProposerWhiteList**:
    - **proposers**: proposer whitelist.

Returns

- **Address**: organization address.

GetProposal

```
rpc GetProposal(aelf.Hash) returns (ProposalOutput) { }

message ProposalOutput {
    aelf.Hash proposal_id = 1;
    string contract_method_name = 2;
    aelf.Address to_address = 3;
    bytes params = 4;
    google.protobuf.Timestamp expired_time = 5;
    aelf.Address organization_address = 6;
    aelf.Address proposer = 7;
    bool to_be_released = 8;
    int64 approval_count = 9;
    int64 rejection_count = 10;
    int64 abstention_count = 11;
}
```

Get the proposal with the given id.

- **Hash**: proposal id.

Returns

- **ProposalOutput**
  - **proposal id**: id of the proposal.
  - **method name**: the method that this proposal will call when being released.
  - **to address**: the address of the target contract.
  - **params**: the parameters of the release transaction.
  - **expiration**: the date at which this proposal will expire.
  - **organization address**: address of this proposals organization.
  - **proposer**: address of the proposer of this proposal.
  - **to be release**: indicates if this proposal is releasable.
  - **approval count**: approval count for this proposal
  - **rejection count**: rejection count for this proposal
  - **abstention count**: abstention count for this proposal

ValidateProposerInWhiteList

```
rpc ValidateProposerInWhiteList(ValidateProposerInWhiteListInput) returns (google.
↪protobuf.BoolValue) { }

message ValidateProposerInWhiteListInput {
    aelf.Address proposer = 1;
    aelf.Address organization_address = 2;
}
```

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**
  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.

Returns

- **BoolValue**: indicates whether the proposer is whitelisted.

2.2 Referendum Contract

2.2.1 CreateOrganization

```
rpc CreateOrganization (CreateOrganizationInput) returns (aelf.Address) { }

message CreateOrganizationInput {
    string token_symbol = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}
```

(continues on next page)

(continued from previous page)

```
message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates an organization and returns its address.

- **CreateOrganizationInput**
  - **token symbol**: the token used during proposal operations.
  - **ProposalReleaseThreshold**:
    - **minimal approval threshold**: the minimum locked token amount threshold for approval.
    - **maximal rejection threshold**: the maximal locked token amount threshold for rejection.
    - **maximal abstention threshold**: the maximal locked token amount threshold for approval.
    - **minimal vote threshold**: the minimum locked token amount threshold for all votes.
  - **ProposerWhiteList**:
    - **proposers**: proposer white list.

Returns

- **Address**: newly created organization address.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization

2.2.2 CreateOrganizationBySystemContract

```
rpc CreateOrganizationBySystemContract(CreateOrganizationBySystemContractInput) returns␣
↪(aelf.Address){}

message CreateOrganizationBySystemContractInput {
    CreateOrganizationInput organization_creation_input = 1;
    string organization_address_feedback_method = 2;
}

message CreateOrganizationInput {
    string token_symbol = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}

message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates an organization by system contract and returns its address. Event **OrganizationCreated** will be fired.

- **CreateOrganizationBySystemContractInput**:
  - **CreateOrganizationInput**:
    - **token symbol**: the token used during proposal operations.
    - **ProposalReleaseThreshold**:
      - **minimal approval threshold**: the minimum locked token amount threshold for approval.
      - **maximal rejection threshold**: the maximal locked token amount threshold for rejection.
      - **maximal abstention threshold**: the maximal locked token amount threshold for approval.
      - **minimal vote threshold**: the minimum locked token amount threshold for all votes.
    - **ProposerWhiteList**:
      - **proposers**: proposer white list.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

Returns

- **Address**: newly created organization address.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization

2.2.3 ReclaimVoteToken

```
rpc ReclaimVoteToken (aelf.Hash) returns (google.protobuf.Empty) { }
```

Used to unlock the tokens that where used for voting.

- **Hash**: proposal id.

2.2.4 ACS3 specific methods

CreateProposal

```
rpc CreateProposal (CreateProposalInput) returns (aelf.Hash) { }

message CreateProposalInput {
    string contract_method_name = 1;
    aelf.Address to_address = 2;
    bytes params = 3;
    google.protobuf.Timestamp expired_time = 4;
```

(continues on next page)

(continued from previous page)

```
    aelf.Address organization_address = 5;
    string proposal_description_url = 6,
    aelf.Hash token = 7;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

This method creates a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract.

- **CreateProposalInput**:
  - **contract method name**: the name of the method to call after release.
  - **to address**: the address of the contract to call after release.
  - **expiration**: the timestamp at which this proposal will expire.
  - **organization address**: the address of the organization.
  - **proposal_description_url**: the url is used for proposal describing.
  - **token**: the token is for proposal id generation and with this token, proposal id can be calculated before proposing.

Returs

- **Hash**: id of the newly created proposal.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

Events

- **ProposalCreated**
  - **proposal_id**: id of the created proposal.

Approve

```
    rpc Approve (aelf.Hash) returns (google.protobuf.Empty) {}

    message ReferendumReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
        aelf.Address address = 2;
        string symbol = 3;
        int64 amount = 4;
        string receipt_type = 5;
        google.protobuf.Timestamp time = 6;
    }
```

This method is called to approve the specified proposal. The amount of token allowance to the proposal virtual address would be locked for voting.

- **Hash**: id of the proposal.

After a successful execution, a **ReferendumReceiptCreated** event log can be found in the transaction result.

Events

- **ReferendumReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: voter address.
  - **token symbol** symbol of token locked.
  - **token amount** amount of token locked.
  - **receipt type**: Approve.
  - **time**: timestamp of this method call.

Reject

```
rpc Reject(aelf.Hash) returns (google.protobuf.Empty) { }

message ReferendumReceiptCreated {
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
    aelf.Address address = 2;
    string symbol = 3;
    int64 amount = 4;
    string receipt_type = 5;
    google.protobuf.Timestamp time = 6;
}
```

This method is called to reject the specified proposal. The amount of token allowance to the proposal virtual address would be locked for voting.

- **Hash**: id of the proposal.

After a successful execution, a **ReferendumReceiptCreated** event log can be found in the transaction result.

Events

- **ReferendumReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: voter address.
  - **token symbol** symbol of token locked.
  - **token amount** amount of token locked.
  - **receipt type**: Reject.
  - **time**: timestamp of this method call.

Abstain

```
    rpc Abstain(aelf.Hash) returns (google.protobuf.Empty) { }

    message ReferendumReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
        aelf.Address address = 2;
        string symbol = 3;
        int64 amount = 4;
        string receipt_type = 5;
        google.protobuf.Timestamp time = 6;
    }
```

This method is called to abstain from the specified proposal. The amount of token allowance to the proposal virtual address would be locked for voting.

- **Hash**: id of the proposal.

After a successful execution, a **ReferendumReceiptCreated** event log can be found in the transaction result.

Events

- **ReferendumReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: voter address.
  - **token symbol** symbol of token locked.
  - **token amount** amount of token locked.
  - **receipt type**: Abstain.
  - **time**: timestamp of this method call.

Release

```
    rpc Release(aelf.Hash) returns (google.protobuf.Empty) { }
```

This method is called to release the specified proposal.

-**Hash**: id of the proposal.

After a successful execution, a **ProposalReleased** event log can be found in the transaction result.

Events

- **ProposalReleased**
  - **proposal id**: id of the proposal.

ChangeOrganizationThreshold

```
rpc ChangeOrganizationThreshold(ProposalReleaseThreshold) returns (google.protobuf.
↪Empty) { }

message ProposalReleaseThreshold {
```

(continues on next page)

(continued from previous page)

```
    int64 minimal_approval_threshold = 1;
    int64 maximal_rejection_threshold = 2;
    int64 maximal_abstention_threshold = 3;
    int64 minimal_vote_threshold = 4;
}

message OrganizationThresholdChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposalReleaseThreshold proposer_release_threshold = 2;
}
```

This method changes the thresholds associated with proposals. All fields will be overwritten by the input value and this will affect all current proposals of the organization. Note: only the organization can execute this through a proposal.

- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the minimum locked token amount threshold for approval.
  - **maximal rejection threshold**: the maximal locked token amount threshold for rejection.
  - **maximal abstention threshold**: the maximal locked token amount threshold for approval.
  - **minimal vote threshold**: the minimum locked token amount threshold for all votes.

After a successful execution, an **OrganizationThresholdChanged** event log can be found in the transaction result.

Events

- **OrganizationThresholdChanged**
  - **organization_address**: the organization address.
  - **proposer_release_threshold**: the new release threshold.

ChangeOrganizationProposerWhiteList

```
rpc ChangeOrganizationProposerWhiteList(ProposerWhiteList) returns (google.protobuf.
↪Empty) { }

message ProposerWhiteList {
    repeated aelf.Address proposers = 1;
}

message OrganizationWhiteListChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposerWhiteList proposer_white_list = 2;
}
```

This method overrides the list of whitelisted proposers.

- **ProposerWhiteList**:
  - **proposers**: the new value for the proposer whitelist.

After a successful execution, a **OrganizationWhiteListChanged** event log can be found in the transaction result.

Events

- **OrganizationWhiteListChanged**
  - **organization_address**: the organization address.
  - **proposer_white_list**: the new proposer whitelist.

CreateProposalBySystemContract

```
rpc CreateProposalBySystemContract(CreateProposalBySystemContractInput) returns (aelf.
↪Hash) { }

message CreateProposalBySystemContractInput {
    acs3.CreateProposalInput proposal_input = 1;
    aelf.Address origin_proposer = 2;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**:
  - **CreateProposalInput**:
    - **contract method name**: the name of the method to call after release.
    - **to address**: the address of the contract to call after release.
    - **expiration**: the date at which this proposal will expire.
    - **organization address**: the address of the organization.
    - **proposal_description_url**: the url is used for proposal describing.
    - **token**: the token is for proposal id generation and proposal id can be calculated before proposing.
  - **origin proposer**: the actor that trigger the call.

Returns

- **Address**: newly created organization address.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

Events

- **ProposalCreated**:
  - **proposal_id**: id of the created proposal.

ClearProposal

```
rpc ClearProposal(aelf.Hash) returns (google.protobuf.Empty) { }
```

Removes the specified proposal.

- **Hash**: id of the proposal to be cleared.

ValidateOrganizationExist

```
rpc ValidateOrganizationExist(aelf.Address) returns (google.protobuf.BoolValue) { }
```

Checks the existence of an organization.

- **Address**: organization address to be checked.

Returns

- **BoolValue**: indicates whether the organization exists.

2.2.5 View methods

GetOrganization

```
rpc GetOrganization (aelf.Address) returns (Organization) { }

message Organization {
    acs3.ProposalReleaseThreshold proposal_release_threshold = 1;
    string token_symbol = 2;
    aelf.Address organization_address = 3;
    aelf.Hash organization_hash = 4;
    acs3.ProposerWhiteList proposer_white_list = 5;
}
```

Returns the organization with the provided organization address.

- **Address**: organization address.

Returns

- **Organization**
  - **ProposalReleaseThreshold**
    - **minimal approval threshold**: the minimum locked token amount threshold for approval.
    - **maximal rejection threshold**: the maximal locked token amount threshold for rejection.
    - **maximal abstention threshold**: the maximal locked token amount threshold for approval.
    - **minimal vote threshold**: the minimum locked token amount threshold for all votes.
  - **token**: token used for proposal operations.
  - **organization address**: organization address.
  - **organization hash**: organization id.
  - **ProposerWhiteList**:

        * **proposers**: proposer white list.

CalculateOrganizationAddress

```
rpc CalculateOrganizationAddress(CreateOrganizationInput) returns (aelf.Address){}

message CreateOrganizationInput {
    string token_symbol = 1;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 2;
    acs3.ProposerWhiteList proposer_white_list = 3;
}
```

Calculates with input and returns the organization address.

- **CreateOrganizationInput**
  - **token symbol**: the token used during proposal operations.
  - **ProposalReleaseThreshold**
    * **minimal approval threshold**: the minimum locked token amount threshold for approval.
    * **maximal rejection threshold**: the maximal locked token amount threshold for rejection.
    * **maximal abstention threshold**: the maximal locked token amount threshold for approval.
    * **minimal vote threshold**: the minimum locked token amount threshold for all votes.
  - **ProposerWhiteList**:
    * **proposers**: proposer white list.

Returns

- **Address**: organization address.

GetProposalVirtualAddress

```
rpc GetProposalVirtualAddress(aelf.Hash) returns (aelf.Address){}
```

Get virtual address for the proposal.

- **Hash**: id of the proposal.

Returns

- **Address**: the virtual address for proposal.

GetProposal

```
rpc GetProposal(aelf.Hash) returns (ProposalOutput) { }

message ProposalOutput {
    aelf.Hash proposal_id = 1;
    string contract_method_name = 2;
    aelf.Address to_address = 3;
    bytes params = 4;
```

(continues on next page)

(continued from previous page)

```
    google.protobuf.Timestamp expired_time = 5;
    aelf.Address organization_address = 6;
    aelf.Address proposer = 7;
    bool to_be_released = 8;
    int64 approval_count = 9;
    int64 rejection_count = 10;
    int64 abstention_count = 11;
}
```

Gets the proposal with the given id.

- **Hash**: proposal id.

Returns

- **ProposalOutput**
  - **proposal id**: id of the proposal.
  - **method name**: the method that this proposal will call when being released.
  - **to address**: the address of the target contract.
  - **params**: the parameters of the release transaction.
  - **expiration**: the date at which this proposal will expire.
  - **organization address**: address of this proposals organization.
  - **proposer**: address of the proposer of this proposal.
  - **to be release**: indicates if this proposal is releasable.
  - **approval count**: locked token amount for approval.
  - **rejection count**: locked token amount for rejection.
  - **abstention count**: locked token amount for abstention.

ValidateProposerInWhiteList

```
rpc ValidateProposerInWhiteList(ValidateProposerInWhiteListInput) returns (google.
↪protobuf.BoolValue) { }

message ValidateProposerInWhiteListInput {
    aelf.Address proposer = 1;
    aelf.Address organization_address = 2;
}
```

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**
  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.

Returns

- **BoolValue**: indicates whether the proposer is whitelisted.

2.3 Parliament Contract

2.3.1 Initialize

```
rpc Initialize(InitializeInput) returns (google.protobuf.Empty) {}

message InitializeInput{
    aelf.Address privileged_proposer = 1;
    bool proposer_authority_required = 2;
}
```

Initialize will set parliament proposer whitelist and create the first parliament organization with specific **proposer_authority_required**.

- **InitializeInput**:
  - **privileged proposer**: privileged proposer would be the first address in parliament proposer whitelist.
  - **proposer authority required**: the setting indicates if proposals need authority to be created for first/default parliament organization.

2.3.2 CreateOrganization

```
rpc CreateOrganization (CreateOrganizationInput) returns (aelf.Address) { }

message CreateOrganizationInput {
    acs3.ProposalReleaseThreshold proposal_release_threshold = 1;
    bool proposer_authority_required = 2;
    bool parliament_member_proposing_allowed = 3;
}

message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates parliament organization with input data.

- **CreateOrganizationInput**
  - **ProposalReleaseThreshold**
    - **minimal approval threshold**: the value to be divided by `10000` for the minimum approval threshold in fraction.
    - **maximal rejection threshold**: the value to be divided by `10000` for the maximal rejection threshold in fraction.
    - **maximal abstention threshold**: the value to be divided by `10000` for the maximal abstention threshold in fraction.
    - **minimal vote threshold**: the value to be divided by `10000` for the minimal vote threshold in fraction.
  - **proposer authority required**: setting this to true can allow anyone to create proposals.

  - **parliament member proposing allowed**: setting this to true can allow parliament member to create proposals.

Returns

- **Address**: the address of newly created organization.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization.

2.3.3 CreateOrganizationBySystemContract

```
rpc CreateOrganizationBySystemContract(CreateOrganizationBySystemContractInput) returns␣
↪(aelf.Address){}

message CreateOrganizationBySystemContractInput {
    CreateOrganizationInput organization_creation_input = 1;
    string organization_address_feedback_method = 2;
}

message OrganizationCreated{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
}
```

Creates parliament organization when called by system contract.

- **CreateOrganizationBySystemContractInput**
  - **CreateOrganizationInput**
    - **ProposalReleaseThreshold**
      - **minimal approval threshold**: the value to be divided by `10000` for the minimum approval threshold in fraction.
      - **maximal rejection threshold**: the value to be divided by `10000` for the maximal rejection threshold in fraction.
      - **maximal abstention threshold**: the value to be divided by `10000` for the maximal abstention threshold in fraction.
      - **minimal vote threshold**: the value to be divided by `10000` for the minimal vote threshold in fraction.
    - **proposer authority required**: setting this to true can allow anyone to create proposals.
    - **parliament member proposing allowed**: setting this to true can allow parliament member to create proposals.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

Returns

- **Address**: the address of newly created organization.

After a successful execution, an **OrganizationCreated** event log can be found in the transaction result.

Events

- **OrganizationCreated**
  - **organization address**: the address of newly created organization.

2.3.4 ACS3 specific methods

CreateProposal

```
rpc CreateProposal (CreateProposalInput) returns (aelf.Hash) { }

message CreateProposalInput {
    string contract_method_name = 1;
    aelf.Address to_address = 2;
    bytes params = 3;
    google.protobuf.Timestamp expired_time = 4;
    aelf.Address organization_address = 5;
    string proposal_description_url = 6,
    aelf.Hash token = 7;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

This method creates a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract.

- **CreateProposalInput**
  - **contract method name**: the name of the method to call after release.
  - **to address**: the address of the contract to call after release.
  - **expiration**: the timestamp at which this proposal will expire.
  - **organization address**: the address of the organization.
  - **proposal_description_url**: the url is used for proposal describing.
  - **token**: the token is for proposal id generation and with this token, proposal id can be calculated before proposing.

Returns

- **Hash**: id of the newly created proposal.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

Events

- **ProposalCreated**
  - **proposal_id**: id of the created proposal.

Approve

```
rpc Approve (aelf.Hash) returns (google.protobuf.Empty) {}

message ReceiptCreated {
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
    aelf.Address address = 2;
    string receipt_type = 3;
    google.protobuf.Timestamp time = 4;
}
```

This method is called to approve the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address who votes for approval.
  - **receipt type**: Approve.
  - **time**: timestamp of this method call.

Reject

```
rpc Reject(aelf.Hash) returns (google.protobuf.Empty) { }

message ReceiptCreated {
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
    aelf.Address address = 2;
    string receipt_type = 3;
    google.protobuf.Timestamp time = 4;
}
```

This method is called to reject the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address who votes for reject.
  - **receipt type**: Reject.
  - **time**: timestamp of this method call.

Abstain

```
    rpc Abstain(aelf.Hash) returns (google.protobuf.Empty) { }

    message ReceiptCreated {
        option (aelf.is_event) = true;
        aelf.Hash proposal_id = 1;
        aelf.Address address = 2;
        string receipt_type = 3;
        google.protobuf.Timestamp time = 4;
    }
```

This method is called to abstain from the specified proposal.

- **Hash**: id of the proposal.

After a successful execution, a **ReceiptCreated** event log can be found in the transaction result.

Events

- **ReceiptCreated**
  - **proposal id**: id of the proposal.
  - **address**: send address who votes for abstention.
  - **receipt type**: Abstain.
  - **time**: timestamp of this method call.

Release

```
    rpc Release(aelf.Hash) returns (google.protobuf.Empty) { }
```

This method is called to release the specified proposal.

-**Hash**: id of the proposal.

After a successful execution, a **ProposalReleased** event log can be found in the transaction result.

Events

- **ProposalReleased**
  - **proposal id**: id of the proposal.

ChangeOrganizationThreshold

```
rpc ChangeOrganizationThreshold(ProposalReleaseThreshold) returns (google.protobuf.
↪Empty) { }

message ProposalReleaseThreshold {
    int64 minimal_approval_threshold = 1;
    int64 maximal_rejection_threshold = 2;
    int64 maximal_abstention_threshold = 3;
    int64 minimal_vote_threshold = 4;
}
```

(continues on next page)

(continued from previous page)

```
message OrganizationThresholdChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposalReleaseThreshold proposer_release_threshold = 2;
}
```

This method changes the thresholds associated with proposals. All fields will be overwritten by the input value and this will affect all current proposals of the organization. Note: only the organization can execute this through a proposal.

- **ProposalReleaseThreshold**:
  - **minimal approval threshold**: the new value for the minimum approval threshold.
  - **maximal rejection threshold**: the new value for the maximal rejection threshold.
  - **maximal abstention threshold**: the new value for the maximal abstention threshold.
  - **minimal vote threshold**: the new value for the minimal vote threshold.

After a successful execution, an **OrganizationThresholdChanged** event log can be found in the transaction result.

Events

- **OrganizationThresholdChanged**
  - **organization_address**: the organization address.
  - **proposer_release_threshold**: the new release threshold.

ChangeOrganizationProposerWhiteList

```
rpc ChangeOrganizationProposerWhiteList(ProposerWhiteList) returns (google.protobuf.
↪Empty) { }

message ProposerWhiteList {
    repeated aelf.Address proposers = 1;
}

message OrganizationWhiteListChanged{
    option (aelf.is_event) = true;
    aelf.Address organization_address = 1;
    ProposerWhiteList proposer_white_list = 2;
}
```

This method overrides the list of whitelisted proposers.

- **ProposerWhiteList**:
  - **proposers**: the new value for the list.

After a successful execution, an **OrganizationWhiteListChanged** event log can be found in the transaction result.

Events

- **OrganizationWhiteListChanged**

  - **organization_address**: the organization address.
  - **proposer_white_list**: the new proposer whitelist.

CreateProposalBySystemContract

```
rpc CreateProposalBySystemContract(CreateProposalBySystemContractInput) returns (aelf.
↪Hash) { }

message CreateProposalBySystemContractInput {
    acs3.CreateProposalInput proposal_input = 1;
    aelf.Address origin_proposer = 2;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**:
  - **CreateProposalInput**:
    * **contract method name**: the name of the method to call after release.
    * **to address**: the address of the contract to call after release.
    * **expiration**: the date at which this proposal will expire.
    * **organization address**: the address of the organization.
    * **proposal_description_url**: the url is used for proposal describing.
    * **token**: the token is for proposal id generation and proposal id can be calculated before proposing.
  - **origin proposer**: the actor that trigger the call.

Returns

- **Address**: id of the newly created proposal.

After a successful execution, an **OrganizationWhiteListChanged** event log can be found in the transaction result.

Events

- **ProposalCreated**:
  - **proposal_id**: id of the created proposal.

ClearProposal

```
rpc ClearProposal(aelf.Hash) returns (google.protobuf.Empty) { }
```

Removes the specified proposal.

- **Hash**: id of the proposal to be cleared.

ValidateOrganizationExist

```
rpc ValidateOrganizationExist(aelf.Address) returns (google.protobuf.BoolValue) { }
```

Checks the existence of an organization.

- **Address**: organization address to be checked.

Returns

- **BoolValue**: indicates whether the organization exists.

2.3.5 View methods

GetOrganization

```
rpc GetOrganization (aelf.Address) returns (Organization) { }

message Organization {
    bool proposer_authority_required = 1;
    aelf.Address organization_address = 2;
    aelf.Hash organization_hash = 3;
    acs3.ProposalReleaseThreshold proposal_release_threshold = 4;
    bool parliament_member_proposing_allowed = 5;
}
```

Returns the organization with the provided organization address.

- **Address**: organization address.

Returns

- **Organization**
  - **proposer authority required**: indicates if proposals need authority to be created.
  - **organization_address**: organization address.
  - **organization hash**: organization id.
  - **ProposalReleaseThreshold**:
    - **minimal approval threshold**: the value to be divided by `10000` for the minimum approval threshold in fraction.
    - **maximal rejection threshold**: the value to be divided by `10000` for the maximal rejection threshold in fraction.
    - **maximal abstention threshold**: the value to be divided by `10000` for the maximal abstention threshold in fraction.
    - **minimal vote threshold**: the value to be divided by `10000` for the minimal vote threshold in fraction.
  - **parliament member proposing allowed** indicates if parliament member can propose to this organization.

GetDefaultOrganizationAddress

```
rpc GetDefaultOrganizationAddress (google.protobuf.Empty) returns (aelf.Address) { }
```

Returns

- **Address**: the address of the default organization.

ValidateAddressIsParliamentMember

```
rpc ValidateAddressIsParliamentMember(aelf.Address) returns (google.protobuf.BoolValue)
↪{ }
```

Validates if the provided address is a parliament member.

- **Address**: parliament member address to be checked.

Returns

- **BoolValue**: indicates whether provided address is one of parliament members.

GetProposerWhiteList

```
rpc GetProposerWhiteList(google.protobuf.Empty) returns (acs3.ProposerWhiteList) { }

message ProposerWhiteList {
    repeated aelf.Address proposers = 1;
}
```

Returns a list of whitelisted proposers.

Returns

- **ProposerWhiteList**
  - **proposers**: the whitelisted proposers.

GetNotVotedPendingProposals

```
rpc GetNotVotedPendingProposals(ProposalIdList) returns (ProposalIdList) { }
message ProposalIdList{
    repeated aelf.Hash proposal_ids = 1;
}
```

Filter still pending ones not yet voted by the `sender` from provided proposals.

- **ProposalIdList**
  - **proposal ids**: list of proposal id.

Returns

- **ProposalIdList**
  - **proposal ids**: filtered proposal id list from input ones.

GetNotVotedProposals

```
rpc GetNotVotedProposals(ProposalIdList) returns (ProposalIdList) { }
message ProposalIdList{
    repeated aelf.Hash proposal_ids = 1;
}
```

Filter not yet voted ones by the `sender` from provided proposals.

- **ProposalIdList**
  - **proposal ids**: list of proposal id.

Returns

- **ProposalIdList**
  - **proposal ids**: filtered proposal id list from input ones.

CalculateOrganizationAddress

```
rpc CalculateOrganizationAddress(CreateOrganizationInput) returns (aelf.Address){}

message CreateOrganizationInput {
    acs3.ProposalReleaseThreshold proposal_release_threshold = 1;
    bool proposer_authority_required = 2;
    bool parliament_member_proposing_allowed = 3;
}
```

Calculates with input and returns the organization address.

- **CreateOrganizationInput**
  - **ProposalReleaseThreshold**
    - **minimal approval threshold**: the value to be divided by `10000` for the minimum approval threshold in fraction.
    - **maximal rejection threshold**: the value to be divided by `10000` for the maximal rejection threshold in fraction.
    - **maximal abstention threshold**: the value to be divided by `10000` for the maximal abstention threshold in fraction.
    - **minimal vote threshold**: the value to be divided by `10000` for the minimal vote threshold in fraction.
  - **proposer authority required**: setting this to true can allow anyone to create proposals.
  - **parliament member proposing allowed**: setting this to true can allow parliament member to create proposals.

Returns

- **Address**: organization address.

GetProposal

```
rpc GetProposal(aelf.Hash) returns (ProposalOutput) { }

message ProposalOutput {
    aelf.Hash proposal_id = 1;
    string contract_method_name = 2;
    aelf.Address to_address = 3;
    bytes params = 4;
    google.protobuf.Timestamp expired_time = 5;
    aelf.Address organization_address = 6;
    aelf.Address proposer = 7;
    bool to_be_released = 8;
    int64 approval_count = 9;
    int64 rejection_count = 10;
    int64 abstention_count = 11;
}
```

Get the proposal with the given id.

- **Hash**: proposal id.

Returns

- **ProposalOutput**
  - **proposal id**: id of the proposal.
  - **method name**: the method that this proposal will call when being released.
  - **to address**: the address of the target contract.
  - **params**: the parameters of the release transaction.
  - **expiration**: the date at which this proposal will expire.
  - **organization address**: address of this proposals organization.
  - **proposer**: address of the proposer of this proposal.
  - **to be release**: indicates if this proposal is releasable.
  - **approval count**: approval count for this proposal.
  - **rejection count**: rejection count for this proposal.
  - **abstention count**: abstention count for this proposal.

ValidateProposerInWhiteList

```
rpc ValidateProposerInWhiteList(ValidateProposerInWhiteListInput) returns (google.
↪protobuf.BoolValue) { }

message ValidateProposerInWhiteListInput {
    aelf.Address proposer = 1;
    aelf.Address organization_address = 2;
}
```

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**
  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.

Returns

- **BoolValue**: indicates whether the proposer is whitelisted.

2.4 Consensus Contract

The Consensus contract is essentially used for managing block producers and synchronizing data.

2.4.1 view methods

For reference, you can find here the available view methods.

GetCurrentMinerList

Gets the list of current miners.

```
rpc GetCurrentMinerList (google.protobuf.Empty) returns (MinerList){
}

message MinerList {
    repeated bytes pubkeys = 1;
}
```

returns:

- **pubkeys**: miners' public keys.

GetCurrentMinerPubkeyList

Gets the list of current miners, each item a block producer's public key in hexadecimal format.

```
 rpc GetCurrentMinerPubkeyList (google.protobuf.Empty) returns (PubkeyList){
 }

message PubkeyList {
    repeated string pubkeys = 1;
}
```

returns:

- **pubkeys**: miner's public key (hexadecimal string).

GetCurrentMinerListWithRoundNumber

Gets the list of current miners along with the round number.

```
rpc GetCurrentMinerListWithRoundNumber (google.protobuf.Empty) returns␣
↪(MinerListWithRoundNumber){
}

message MinerListWithRoundNumber {
    MinerList miner_list = 1;
    sint64 round_number = 2;
}

message MinerList {
    repeated bytes pubkeys = 1;
}
```

returns:

- **miner list**: miners list.
- **round number**: current round number.

MinerList:

- **pubkeys**: miners' public keys.

GetRoundInformation

Gets information of the round specified as input.

```
rpc GetRoundInformation (aelf.SInt64Value) returns (Round){
}

message SInt64Value
{
    sint64 value = 1;
}

message Round {
    sint64 round_number = 1;
    map<string, MinerInRound> real time miners information = 2;
    sint64 main chain miners round number = 3;
    sint64 blockchain age = 4;
    string extra block producer of previous round = 7;
    sint64 term number = 8;
    sint64 confirmed irreversible block height = 9;
    sint64 confirmed irreversible block round number = 10;
    bool is miner list just changed = 11;
    sint64 round id for validation = 12;
}

message MinerInRound {
    sint32 order = 1;
    bool is extra block producer = 2;
    aelf.Hash in value = 3;
    aelf.Hash out value = 4;
    aelf.Hash signature = 5;
```

(continues on next page)

(continued from previous page)

```
    google.protobuf.Timestamp expected mining time = 6;
    sint64 produced blocks = 7;
    sint64 missed time slots = 8;
    string pubkey = 9;
    aelf.Hash previous in value = 12;
    sint32 supposed order of next round = 13;
    sint32 final order of next round = 14;
    repeated google.protobuf.Timestamp actual mining times = 15;
    map<string, bytes> encrypted pieces = 16;
    map<string, bytes> decrypted pieces = 17;
    sint32 produced tiny blocks = 18;
    sint64 implied irreversible block height = 19;
}
```

SInt64Value:

- **value**: round number.

returns:

- **round number**: round number.
- **real time miners information**: public key => miner information.
- **blockchain age**: current time minus block chain start time (if the round number is 1, the block chain age is 1), represented in seconds.
- **extra block producer of previous round**: the public key (hexadecimal string) of the first miner, who comes from the last term, in the current term.
- **term number**: the current term number.
- **confirmed irreversible block height**: irreversible block height.
- **confirmed irreversible block round number**: irreversible block round number.
- **is miner list just changed**: is miner list different from the the miner list in the previous round.
- **round id for validation**: round id, calculated by summing block producers' expecting time (second).

MinerInRound:

- **order**: the order of miners producing block.
- **is extra block producer**: The miner who is the first miner in the first round of each term.
- **in value**: the previous miner's public key.
- **out value**: the post miner's public key.
- **signature**: self signature.
- **expected mining time**: expected mining time.
- **produced blocks**: produced blocks.
- **missed time slots**: missed time slots.
- **pubkey**: public key string.
- **previous in value**: previous miner's public key.
- **supposed order of next round**: evaluated order in next round.
- **final order of next round**: the real order in the next round.

- **actual mining times**: the real mining time.
- **encrypted pieces**: public key (miners in the current round) => message encrypted by shares information and public key (represented by hexadecimal string).
- **decrypted pieces**: the message of miners in the previous round.
- **produced tiny blocks**: produced tiny blocks.
- **implied irreversible block height**: miner records a irreversible block height.

GetCurrentRoundNumber

Gets the current round number.

```
rpc GetCurrentRoundNumber (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: number of current round.

GetCurrentRoundInformation

Gets the current round's information.

```
rpc GetCurrentRoundInformation (google.protobuf.Empty) returns (Round){
}
```

returns:

- **round number**: round number.
- **real time miners information**: public key => miner information.
- **main chain miners round number**: is not used.
- **blockchain age**: current time minus block chain start time stamp (if the round number is 1, the block chain age is 1), represented by second.
- **extra block producer of previous round**: the public key (hexadecimal string) of the first miner, who comes from the last term, in the current term.
- **term number**: the current term number.
- **confirmed irreversible block height**: irreversible block height.
- **confirmed irreversible block round number**: irreversible block round number.
- **is miner list just changed**: is miner list different from the the miner list in the previous round.
- **round id for validation**: round id, calculated by summing block producers' expecting time(second).

MinerInRound:

- **order**: the order of miners producing block.

- **is extra block producer**: The miner who is the first miner in the first round of each term.
- **in value**: the previous miner's public key.
- **out value**: the post miner's public key.
- **signature**: self signature.
- **expected mining time**: expected mining time.
- **produced blocks**: produced blocks.
- **missed time slots**: missed time slots.
- **pubkey**: public key string.
- **previous in value**: previous miner's previous miner's public key.
- **supposed order of next round**: evaluated order in next round.
- **final order of next round**: the real order in the next round.
- **actual mining times**: the real mining time.
- **encrypted pieces**: public key (miners in the current round) => message encrypted by shares information and public key(represented by hexadecimal string).
- **decrypted pieces**: the message of miners in the previous round.
- **produced tiny blocks**: produced tiny blocks.
- **implied irreversible block height**: miner records a irreversible block height.

GetPreviousRoundInformation

Gets the previous round information.

```
rpc GetPreviousRoundInformation (google.protobuf.Empty) returns (Round){
}
```

returns:

- **round number**: round number.
- **real time miners information**: public key => miner information.
- **main chain miners round number**: is not used.
- **blockchain age**: current time minus block chain start time stamp (if the round number is 1, the block chain age is 1), represented by second.
- **extra block producer of previous round**: the public key(hexadecimal string) of the first miner, who comes from the last term, in the current term.
- **term number**: the current term number.
- **confirmed irreversible block height**: irreversible block height.
- **confirmed irreversible block round number**: irreversible block round number.
- **is miner list just changed**: is miner list different from the the miner list in the previous round.
- **round id for validation**: round id, calculated by summing block producers' expecting time(second).

MinerInRound:

- **order**: the order of miners producing block.

- **is extra block producer**: The miner who is the first miner in the first round of each term.
- **in value**: the previous miner's public key.
- **out value**: the post miner's public key.
- **signature**: self signature.
- **expected mining time**: expected mining time.
- **produced blocks**: produced blocks.
- **missed time slots**: missed time slots.
- **pubkey**: public key string.
- **previous in value**: previous miner's previous miner's public key.
- **supposed order of next round**: evaluated order in next round.
- **final order of next round**: the real order in the next round.
- **actual mining times**: the real mining time.
- **encrypted pieces**: public key (miners in the current round) => message encrypted by shares information and public key(represented by hexadecimal string).
- **decrypted pieces**: the message of miners in the previous round.
- **produced tiny blocks**: produced tiny blocks.
- **implied irreversible block height**: miner records a irreversible block height.

GetCurrentTermNumber

Gets the current term number.

```
rpc GetCurrentTermNumber (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: the current term number.

GetCurrentWelfareReward

Gets the current welfare reward.

```
rpc GetCurrentWelfareReward (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: the current welfare reward.

GetPreviousMinerList

Gets the miners in the previous term.

```
rpc GetPreviousMinerList (google.protobuf.Empty) returns (MinerList){
}

message MinerList {
    repeated bytes pubkeys = 1;
}
```

MinerList:

- **pubkeys**: public keys (represented by hexadecimal strings) of miners in the previous term.

GetMinedBlocksOfPreviousTerm

Gets the number of mined blocks during the previous term.

```
rpc GetMinedBlocksOfPreviousTerm (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: the number of mined blocks.

GetNextMinerPubkey

Gets the miner who will produce the block next, which means the miner is the first one whose expected mining time is greater than the current time. If this miner can not be found, the first miner who is extra block producer will be selected.

```
rpc GetNextMinerPubkey (google.protobuf.Empty) returns (google.protobuf.StringValue){
}

message StringValue {
  string value = 1;
}
```

returns:

- **value**: the miner's public key.

GetCurrentMinerPubkey

Gets the current miner.

```
rpc GetCurrentMinerPubkey (google.protobuf.Empty) returns (google.protobuf.StringValue){
}

message StringValue {
  string value = 1;
}
```

returns:

- **value**: miner's public key.

IsCurrentMiner

Query whether the miner is the current miner.

```
rpc IsCurrentMiner (aelf.Address) returns (google.protobuf.BoolValue){
}

message Address
{
    bytes value = 1;
}
```

Address:

- **value**: miner's address.

returns:

- **value**: indicates if the input miner is the current miner.

GetNextElectCountDown

Count down to the next election.

```
rpc GetNextElectCountDown (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: total seconds to next election.

2.5 Election Contract

The Election contract is essentially used for voting for Block Producers.

2.5.1 AnnounceElection

To be a block producer, a user should first register to be a candidate and lock some token as a deposit. If the data center is not full, the user will be added in automatically and get one weight (10 weight limited) for sharing bonus in the future.

```
rpc AnnounceElection (google.protobuf.Empty) returns (google.protobuf.Empty){
}
```

2.5.2 QuitElection

A candidate is able to quit the election provided he is not currently elected. If you quit successfully, the candidate will get his locked tokens back and will not receive anymore bonus.

```
rpc QuitElection (google.protobuf.Empty) returns (google.protobuf.Empty){
}
```

2.5.3 Vote

Used for voting for a candidate to be elected. The tokens you vote with will be locked until the end time. According to the number of token you voted and its lock time, you can get corresponding weight for sharing the bonus in the future.

```
rpc Vote (VoteMinerInput) returns (aelf.Hash){
}

message VoteMinerInput {
    string candidate_pubkey = 1;
    sint64 amount = 2;
    google.protobuf.Timestamp end_timestamp = 3;
}

message Hash
{
    bytes value = 1;
}
```

VoteMinerInput:

- **candidate pubkey**: candidate public key.
- **amount**: amount token to vote.
- **end timestamp**: before which, your vote works.

returns:

- **value**: vote id.

2.5.4 ChangeVotingOption

Before the end time, you are able to change your vote target to other candidates.

```
rpc ChangeVotingOption (ChangeVotingOptionInput) returns google.protobuf.Empty){
}

message ChangeVotingOptionInput {
    aelf.Hash vote_id = 1;
    string candidate_pubkey = 2;
}
```

ChangeVotingOptionInput:

- **voting vote id**: transaction id.
- **candidate pubkey**: new candidate public key.

2.5.5 Withdraw

After the lock time, your locked tokens will be unlocked and you can withdraw them.

```
rpc Withdraw (aelf.Hash) returns (google.protobuf.Empty){
}

message Hash
{
    bytes value = 1;
}
```

Hash:

- **value**: transaction id.

2.5.6 SetVoteWeightProportion

Vote weight calcualtion takes in consideration the amount you vote and the lock time your vote.

```
rpc SetVoteWeightProportion (VoteWeightProportion) returns (google.protobuf.Empty){
}

message VoteWeightProportion {
    int32 time_proportion = 1;
    int32 amount_proportion = 2;
}
```

VoteWeightProportion:

- **time proportion**: time's weight.
- **amount proportion**: amount's weight.

2.5.7 view methods

For reference, you can find here the available view methods.

GetCandidates

Gets all candidates' public keys.

```
rpc GetCandidates (google.protobuf.Empty) returns (PubkeyList){
}

message PubkeyList {
    repeated bytes value = 1;
}
```

returns:

- **value** public key array of candidates

GetVotedCandidates

Gets all candidates whose number of votes is greater than 0.

```
rpc GetVotedCandidates (google.protobuf.Empty) returns (PubkeyList){
}

message PubkeyList {
    repeated bytes value = 1;
}
```

returns:

- **value** public key array of candidates.

GetCandidateInformation

Gets a candidate's information. If the candidate does not exist, it will return a candidate without any information.

```
rpc GetCandidateInformation (google.protobuf.StringValue) returns (CandidateInformation){
}

message StringValue {
  string value = 1;
}

message CandidateInformation {
    string pubkey = 1;
    repeated sint64 terms = 2;
    sint64 produced_blocks = 3;
    sint64 missed_time_slots = 4;
    sint64 continual_appointment_count = 5;
    aelf.Hash announcement_transaction_id = 6;
    bool is_current_candidate = 7;
}
```

StringValue:

- **value**: public key (hexadecimal string) of the candidate.

returns:

- **pubkey**: public key (represented by an hexadecimal string).
- **terms**: indicates in which terms the candidate participated.
- **produced blocks**: the number of blocks the candidate has produced.
- **missed time slots**: the time slot for which the candidate failed to produce blocks.
- **continual appointment count**: the time the candidate continue to participate in the election.
- **announcement transaction id**: the transaction id that the candidate announce.
- **is current candidate**: indicate whether the candidate can be elected in the current term.

GetVictories

Gets the victories of the latest term.

```
rpc GetVictories (google.protobuf.Empty) returns (PubkeyList){
}

message PubkeyList {
    repeated bytes value = 1;
}
```

returns:

- **value** the array of public key who has been elected as block producers.

GetTermSnapshot

Gets the snapshot of the term provided as input.

```
rpc GetTermSnapshot (GetTermSnapshotInput) returns (TermSnapshot){
}

message GetTermSnapshotInput {
    sint64 term_number = 1;
}

message TermSnapshot {
    sint64 end_round_number = 1;
    sint64 mined_blocks = 2;
    map<string, sint64> election_result = 3;
}
```

GetTermSnapshotInput:

- **term number**: term number.

returns:

- **end round number**: the last term id be saved.
- **mined blocks**: number of blocks produced in previous term.
- **election result**: candidate => votes.

GetMinersCount

Count miners.

```
rpc GetMinersCount (google.protobuf.Empty) returns (aelf.SInt32Value){
}

message SInt32Value
{
    sint32 value = 1;
}
```

returns:

- **value**: the total number of block producers.

GetElectionResult

Gets an election result by term id.

```
rpc GetElectionResult (GetElectionResultInput) returns (ElectionResult){
}

message GetElectionResultInput {
    sint64 term_number = 1;
}

message ElectionResult {
    sint64 term_number = 1;
    map<string, sint64> results = 2;
    bool is_active = 3;
}
```

GetElectionResultInput:

- **term number**: term id.

returns:

- **term number**: term id.
- **results**: candidate => votes.
- **is active**: indicates that if the term number you input is the current term.

GetElectorVote

Gets the voter's information.

```
rpc GetElectorVote (google.protobuf.StringValue) returns (ElectorVote){
}

message StringValue {
  string value = 1;
}
```

(continues on next page)

(continued from previous page)

```
message ElectorVote {
    repeated aelf.Hash active_voting_record_ids = 1;// Not withdrawn.
    repeated aelf.Hash withdrawn_voting_record_ids = 2;
    sint64 active_voted_votes_amount = 3;
    sint64 all_voted_votes_amount = 4;
    repeated ElectionVotingRecord active_voting_records = 5;
    repeated ElectionVotingRecord withdrawn_votes_records = 6;
    bytes pubkey = 7;
}
```

StringValue:

- **value**: the public key (hexadecimal string) of voter.

returns:

- **active voting record ids**: transaction ids, in which transactions you voted.
- **withdrawn voting record ids**: transaction ids.
- **active voted votes amount**: the number(excluding the withdrawn) of token you vote.
- **all voted votes amount**: the number of token you have voted.
- **active voting records**: no records in this api.
- **withdrawn votes records**: no records in this api.
- **pubkey**: voter public key (byte string).

GetElectorVoteWithRecords

Gets the information about a voter including the votes (excluding withdrawal information).

```
rpc GetElectorVoteWithRecords (google.protobuf.StringValue) returns (ElectorVote){
}

message StringValue {
  string value = 1;
}

message ElectorVote {
    repeated aelf.Hash active_voting_record_ids = 1;// Not withdrawn.
    repeated aelf.Hash withdrawn_voting_record_ids = 2;
    sint64 active_voted_votes_amount = 3;
    sint64 all_voted_votes_amount = 4;
    repeated ElectionVotingRecord active_voting_records = 5;
    repeated ElectionVotingRecord withdrawn_votes_records = 6;
    bytes pubkey = 7;
}

message ElectionVotingRecord {
    aelf.Address voter = 1;
    string candidate = 2;
    sint64 amount = 3;
```

(continues on next page)

(continued from previous page)

```
    sint64 term_number = 4;
    aelf.Hash vote_id = 5;
    sint64 lock_time = 7;
    google.protobuf.Timestamp unlock_timestamp = 10;
    google.protobuf.Timestamp withdraw_timestamp = 11;
    google.protobuf.Timestamp vote_timestamp = 12;
    bool is_withdrawn = 13;
    sint64 weight = 14;
    bool is_change_target = 15;
}
```

StringValue:

- **value**: the public key (hexadecimal string) of the voter.

returns:

- **active voting record ids**: transaction ids, in which transactions you vote.
- **withdrawn voting record ids**: transaction ids.
- **active voted votes amount**: the number(excluding the withdrawn) of token you vote.
- **all voted votes amount**: the number of token you have voted.
- **active voting records**: records of the vote transaction with detail information.
- **withdrawn votes records**: no records in this api.
- **pubkey**: voter public key (byte string).

ElectionVotingRecord:

- **voter**: voter address.
- **candidate**: public key.
- **amount**: vote amount.
- **term number**: snapshot number.
- **vote id**: transaction id.
- **lock time**: time left to unlock token.
- **unlock timestamp**: unlock date.
- **withdraw timestamp**: withdraw date.
- **vote timestamp**: vote date.
- **is withdrawn**: indicates if the vote has been withdrawn.
- **weight**: vote weight for sharing bonus.
- **is change target**: whether vote others.

GetElectorVoteWithAllRecords

Gets the information about a voter including the votes and withdrawal information.

```
rpc GetElectorVoteWithAllRecords (google.protobuf.StringValue) returns (ElectorVote){
}

message StringValue {
  string value = 1;
}

message ElectorVote {
    repeated aelf.Hash active_voting_record_ids = 1;// Not withdrawn.
    repeated aelf.Hash withdrawn_voting_record_ids = 2;
    sint64 active_voted_votes_amount = 3;
    sint64 all_voted_votes_amount = 4;
    repeated ElectionVotingRecord active_voting_records = 5;
    repeated ElectionVotingRecord withdrawn_votes_records = 6;
    bytes pubkey = 7;
}


message ElectionVotingRecord {
    aelf.Address voter = 1;
    string candidate = 2;
    sint64 amount = 3;
    sint64 term_number = 4;
    aelf.Hash vote_id = 5;
    sint64 lock_time = 7;
    google.protobuf.Timestamp unlock_timestamp = 10;
    google.protobuf.Timestamp withdraw_timestamp = 11;
    google.protobuf.Timestamp vote_timestamp = 12;
    bool is_withdrawn = 13;
    sint64 weight = 14;
    bool is_change_target = 15;
}
```

StringValue:

- **value**: the public key (hexadecimal string) of voter.

returns:

- **active voting record ids**: transaction ids, in which transactions you vote.
- **withdrawn voting record ids**: transaction ids.
- **active voted votes amount**: the number(excluding the withdrawn) of token you vote.
- **all voted votes amount**: the number of token you have voted.
- **active voting records**: records of transactions that are active.
- **withdrawn votes records**: records of transactions in which withdraw is true.
- **pubkey**: voter public key (byte string).

ElectionVotingRecord:

- **voter**: voter address.
- **candidate**: public key.
- **amount**: vote amount.

- **term number**: snapshot number.
- **vote id**: transaction id.
- **lock time**: time left to unlock token.
- **unlock timestamp**: unlock date.
- **withdraw timestamp**: withdraw date.
- **vote timestamp**: vote date.
- **is withdrawn**: indicates if the vote has been withdrawn.
- **weight**: vote weight for sharing bonus.
- **is change target**: whether vote others.

GetCandidateVote

Gets statistical information about vote transactions of a candidate.

```
rpc GetCandidateVote (google.protobuf.StringValue) returns (CandidateVote){
}

message StringValue {
  string value = 1;
}

message CandidateVote {
    repeated aelf.Hash obtained_active_voting_record_ids = 1;
    repeated aelf.Hash obtained_withdrawn_voting_record_ids = 2;
    sint64 obtained_active_voted_votes_amount = 3;
    sint64 all_obtained_voted_votes_amount = 4;
    repeated ElectionVotingRecord obtained_active_voting_records = 5;
    repeated ElectionVotingRecord obtained_withdrawn_votes_records = 6;
    bytes pubkey = 7;
}
```

StringValue:

- **value**: public key of the candidate.

returns:

- **obtained active voting record ids**: vote transaction ids.
- **obtained withdrawn voting record ids**: withdrawn transaction ids.
- **obtained active voted votes amount**: the valid number of vote token in current.
- **all obtained voted votes amount**: total number of vote token the candidate has got.
- **obtained active voting records**: no records in this api.
- **obtained withdrawn votes records**: no records in this api.

GetCandidateVoteWithRecords

Gets statistical information about vote transactions of a candidate with the detailed information of the transactions that are not withdrawn.

```
rpc GetCandidateVoteWithRecords (google.protobuf.StringValue) returns (CandidateVote){
}

message StringValue {
  string value = 1;
}

message CandidateVote {
    repeated aelf.Hash obtained_active_voting_record_ids = 1;
    repeated aelf.Hash obtained_withdrawn_voting_record_ids = 2;
    sint64 obtained_active_voted_votes_amount = 3;
    sint64 all_obtained_voted_votes_amount = 4;
    repeated ElectionVotingRecord obtained_active_voting_records = 5;
    repeated ElectionVotingRecord obtained_withdrawn_votes_records = 6;
    bytes pubkey = 7;
}
message ElectionVotingRecord {
    aelf.Address voter = 1;
    string candidate = 2;
    sint64 amount = 3;
    sint64 term_number = 4;
    aelf.Hash vote_id = 5;
    sint64 lock_time = 7;
    google.protobuf.Timestamp unlock_timestamp = 10;
    google.protobuf.Timestamp withdraw_timestamp = 11;
    google.protobuf.Timestamp vote_timestamp = 12;
    bool is_withdrawn = 13;
    sint64 weight = 14;
    bool is_change_target = 15;
}
```

StringValue:

- **value**: public key of the candidate.

returns:

- **obtained active voting record ids**: vote transaction ids.
- **obtained withdrawn voting record ids**: withdraw transaction ids.
- **obtained active voted votes amount**: the valid number of vote token in current.
- **all obtained voted votes amount**: total number of vote token the candidate has got.
- **obtained active voting records**: the records of the transaction without withdrawing.
- **obtained withdrawn votes records**: no records in this api.

ElectionVotingRecord:

- **voter** voter address.
- **candidate** public key.
- **amount** vote amount.
- **term number** snapshot number.
- **vote id** transaction id.

- **lock time** time left to unlock token.
- **unlock timestamp** unlock date.
- **withdraw timestamp** withdraw date.
- **vote timestamp** vote date.
- **is withdrawn** indicates whether the vote has been withdrawn.
- **weight** vote weight for sharing bonus.
- **is change target** whether vote others.

GetCandidateVoteWithAllRecords

Gets statistical information about vote transactions of a candidate with the detailed information of all the transactions.

```
rpc GetCandidateVoteWithAllRecords (google.protobuf.StringValue) returns (CandidateVote){
}

message StringValue {
  string value = 1;
}

message CandidateVote {
    repeated aelf.Hash obtained_active_voting_record_ids = 1;
    repeated aelf.Hash obtained_withdrawn_voting_record_ids = 2;
    sint64 obtained_active_voted_votes_amount = 3;
    sint64 all_obtained_voted_votes_amount = 4;
    repeated ElectionVotingRecord obtained_active_voting_records = 5;
    repeated ElectionVotingRecord obtained_withdrawn_votes_records = 6;
    bytes pubkey = 7;
}

message ElectionVotingRecord {
    aelf.Address voter = 1;
    string candidate = 2;
    sint64 amount = 3;
    sint64 term_number = 4;
    aelf.Hash vote_id = 5;
    sint64 lock_time = 7;
    google.protobuf.Timestamp unlock_timestamp = 10;
    google.protobuf.Timestamp withdraw_timestamp = 11;
    google.protobuf.Timestamp vote_timestamp = 12;
    bool is_withdrawn = 13;
    sint64 weight = 14;
    bool is_change_target = 15;
}
```

StringValue:

- **value**: public key of the candidate.

returns:

- **obtained active voting record ids**: vote transaction ids.

- **obtained withdrawn voting record ids**: withdrawn transaction ids.
- **obtained active voted votes amount**: the valid number of vote token in current.
- **all obtained voted votes amount**: total number of vote token the candidate has got.
- **obtained active voting records**: the records of the transaction without withdrawing.
- **obtained withdrawn votes records**: the records of the transaction withdrawing the vote token.

ElectionVotingRecord:

- **voter**: voter address.
- **candidate**: public key.
- **amount**: vote amount.
- **term number**: snapshot number.
- **vote id**: transaction id.
- **lock time**: time left to unlock token.
- **unlock timestamp**: unlock date.
- **withdraw timestamp**: withdraw date.
- **vote timestamp**: vote date.
- **is withdrawn**: indicates whether the vote has been withdrawn.
- **weight**: vote weight for sharing bonus.
- **is change target**: whether vote others.

GetVotersCount

Gets the total number of voters.

```
rpc GetVotersCount (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: number of voters.

GetVotesAmount

Gets the total number of vote token (not counting those that have been withdrawn).

```
rpc GetVotesAmount (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
```

(continues on next page)

(continued from previous page)

```
    sint64 value = 1;
}
```

returns:

- **value**: number of vote token.

GetCurrentMiningReward

Gets the current block reward (produced block Number times reward unit).

```
rpc GetCurrentMiningReward (google.protobuf.Empty) returns (aelf.SInt64Value){
}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: number of ELF that rewards miner for producing blocks.

GetPageableCandidateInformation

Gets candidates' information according to the page's index and records length.

```
rpc GetPageableCandidateInformation (PageInformation) returns␣
↪(GetPageableCandidateInformationOutput){
}

message PageInformation {
    sint32 start = 1;
    sint32 length = 2;
}

message GetPageableCandidateInformationOutput {
    repeated CandidateDetail value = 1;
}

message CandidateDetail {
    CandidateInformation candidate_information = 1;
    sint64 obtained_votes_amount = 2;
}

message CandidateInformation {
    string pubkey = 1;
    repeated sint64 terms = 2;
    sint64 produced_blocks = 3;
    sint64 missed_time_slots = 4;
    sint64 continual_appointment_count = 5;
    aelf.Hash announcement_transaction_id = 6;
```

(continues on next page)

(continued from previous page)

```
    bool is_current_candidate = 7;
}
```

PageInformation:

- **start**: start index.
- **length**: number of records.

returns:

- **CandidateDetail**: candidates' detailed information.

CandidateDetail:

- **candidate information**: candidate information.
- **obtained votes amount**: obtained votes amount.

CandidateInformation:

- **pubkey**: public key (hexadecimal string).
- **terms**: indicate which terms have the candidate participated.
- **produced blocks**: the number of blocks the candidate has produced.
- **missed time slots**: the time the candidate failed to produce blocks.
- **continual appointment count**: the time the candidate continue to participate in the election.
- **announcement transaction id**: the transaction id that the candidate announce.
- **is current candidate**: indicate whether the candidate can be elected in the current term.

GetMinerElectionVotingItemId

Gets the voting activity id.

```
rpc GetMinerElectionVotingItemId (google.protobuf.Empty) returns (aelf.Hash){
}

message Hash
{
    bytes value = 1;
}
```

returns:

- **value**: voting item id.

GetDataCenterRankingList

Gets the data center ranking list.

```
rpc GetDataCenterRankingList (google.protobuf.Empty) returns (DataCenterRankingList){
}

message DataCenterRankingList {
```

(continues on next page)

(continued from previous page)

```
    map<string, sint64> data_centers = 1;
    sint64 minimum_votes = 2;
}
```

returns:

- **data centers**: the top n * 5 candidates with vote amount.
- **minimum votes**: not be used.

GetVoteWeightProportion

Gets VoteWeight Proportion.

```
rpc GetVoteWeightProportion (google.protobuf.Empty) returns (VoteWeightProportion){
}

message VoteWeightProportion {
    int32 time_proportion = 1;
    int32 amount_proportion = 2;
}
```

note: *for VoteWeightProportion see SetVoteWeightProportion*

GetCalculateVoteWeight

Calculate the concrete vote weight according to your input.

```
rpc GetCalculateVoteWeight (VoteInformation) returns (google.protobuf.Int64Value){
}

message VoteInformation{
    int64 amount = 1;
    int64 lock_time = 2;
}
```

VoteInformation:

- **amount**: the vote amount.
- **lock time**: the lock time your vote.

returns:

- **value**: vote weight calculated with your input and our function.

2.6 Genesis Contract

This page describes available methods on the Genesis Contract.

2.6.1 Method documentation

Views

function CurrentContractSerialNumber

```
rpc CurrentContractSerialNumber (google.protobuf.Empty) returns (google.protobuf.
↪UInt64Value)
{
    option (aelf.is_view) = true;
}
```

Gets the current serial number of genesis contract (corresponds to the serial number that will be given to the next deployed contract).

Parameters:

- **google.protobuf.Empty**

Returns:

Serial number of the genesis contract.

function GetContractInfo

```
rpc GetContractInfo (aelf.Address) returns (ContractInfo)
{
    option (aelf.is_view) = true;
}

message ContractInfo {
    uint64 serial_number = 1;
    aelf.Address author = 2;
    int32 category = 3;
    aelf.Hash code_hash = 4;
    bool is_system_contract = 5;
    int32 version = 6;
}
```

Gets detailed information about the specified contract.

Parameters:

- **Address** - address the contract

Returns:

A **ContractInfo** object that represents detailed information about the specified contract.

function GetContractAuthor

```
rpc GetContractAuthor (aelf.Address) returns (aelf.Address)
{
```

(continues on next page)

(continued from previous page)

```
    option (aelf.is_view) = true;
}
```

Get author of the specified contract.

Parameters:

- **Address** - address of specified contract

Returns:

Author of the specified contract.

function GetContractHash

```
rpc GetContractHash (aelf.Address) returns (aelf.Hash)
{
    option (aelf.is_view) = true;
}
```

Gets the code hash of the contract at the specified address.

Parameters:

- **Address** - address of a contract

Returns:

The code hash of the contract.

function GetContractAddressByName

```
rpc GetContractAddressByName (aelf.Hash) returns (aelf.Address)
{
    option (aelf.is_view) = true;
}
```

Gets the address of a system contract by its name.

Parameters:

- **Hash** - name hash of the contract

Returns:

Address of the specified contract.

function GetSmartContractRegistrationByAddress

```
rpc GetSmartContractRegistrationByAddress (aelf.Address) returns (aelf.
↪SmartContractRegistration)
{
    option (aelf.is_view) = true;
}
```

(continues on next page)

(continued from previous page)

```
message SmartContractRegistration {
    int32 category = 1;
    bytes code = 2;
    Hash code_hash = 3;
    bool is_system_contract = 4;
    int32 version = 5;
}
```

Gets the registration of a smart contract by its address.

Parameters:

- **Address** - address of a smart contract

Returns:

Registration object of the smart contract.

function GetSmartContractRegistrationByAddress

```
rpc GetSmartContractRegistrationByCodeHash (aelf.Hash) returns (aelf.
↪SmartContractRegistration) {
        option (aelf.is_view) = true;
}

message SmartContractRegistration {
    int32 category = 1;
    bytes code = 2;
    Hash code_hash = 3;
    bool is_system_contract = 4;
    int32 version = 5;
}
```

Gets the registration of a smart contract by code hash.

Parameters:

- **Hash** - contract code hash

Returns:

Registration object of the smart contract.

function ValidateSystemContractAddress

```
rpc ValidateSystemContractAddress(ValidateSystemContractAddressInput) returns (google.
↪protobuf.Empty)
{
    option (aelf.is_view) = true;
}

message ValidateSystemContractAddressInput {
```

(continues on next page)

(continued from previous page)

```
    aelf.Hash system_contract_hash_name = 1;
    aelf.Address address = 2;
}
```

Validates whether the input system contract exists.

Parameters:

ValidateSystemContractAddressInput

- **Hash** - name hash of the contract
- **Address** - address of the contract

Returns:

google.protobuf.Empty

Actions

function DeploySystemSmartContract

```
rpc DeploySystemSmartContract (SystemContractDeploymentInput) returns (aelf.Address) {}

message SystemContractDeploymentInput
{
    message SystemTransactionMethodCall
    {
        string method_name = 1;
        bytes params = 2;
    }
    message SystemTransactionMethodCallList
    {
        repeated SystemTransactionMethodCall value = 1;
    }
    sint32 category = 1;
    bytes code = 2;
    aelf.Hash name = 3;
    SystemTransactionMethodCallList transaction_method_call_list = 4;
}
```

Deploys a system smart contract on chain.

Parameters:

SystemContractDeploymentInput

- **category** - contract type
- **code** - byte array of system contract code
- **name** - name hash of system contract
- **transaction_method_call_list** - list of methods called by system transaction

Returns:

Address of the deployed contract.

function ProposeNewContract

```
rpc ProposeNewContract (ContractDeploymentInput) returns (aelf.Hash) {}

message ContractDeploymentInput {
    sint32 category = 1;
    bytes code = 2;
}
```

Propose new contract deployment.

Parameters:

ContractDeploymentInput

- **category** - contract type (usually 0 for now)
- **code** - byte array that represents the contract code

Returns:

Hash of the **ContractDeploymentInput** object.

function ProposeUpdateContract

```
rpc ProposeUpdateContract (ContractUpdateInput) returns (aelf.Hash) {}

message ContractUpdateInput {
    aelf.Address address = 1;
    bytes code = 2;
}
```

Creates a proposal to update the specified contract.

Parameters:

ContractUpdateInput

- **address** - address of the contract to be updated
- **code** - byte array of the contract's new code

Returns:

Hash of the **ContractUpdateInput** object.

function ProposeContractCodeCheck

```
rpc ProposeContractCodeCheck (ContractCodeCheckInput) returns (aelf.Hash) {}

message ContractCodeCheckInput{
    bytes contract_input = 1;
    bool is_contract_deployment = 2;
    string code_check_release_method = 3;
    aelf.Hash proposed_contract_input_hash = 4;
    sint32 category = 5;
}
```

Propose to check the code of a contract.

Parameters:

ContractCodeCheckInput

- **contract_input** - byte array of the contract code to be checked
- **is_contract_deployment** - whether the input contract is to be deployed or updated
- **code_check_release_method** - method to call after code check complete (DeploySmartContract or UpdateSmartContract)
- **proposed_contract_input_hash** - id of the proposed contract
- **category** - contract category (always 0 for now)

Returns:

Hash of the proposed contract.

function ReleaseApprovedContract

```
rpc ReleaseApprovedContract (ReleaseContractInput) returns (google.protobuf.Empty) {}

message ReleaseContractInput {
    aelf.Hash proposal_id = 1;
    aelf.Hash proposed_contract_input_hash = 2;
}
```

Releases a contract proposal which has been approved.

Parameters:

ReleaseContractInput

- **proposal_id** - hash of the proposal
- **proposed_contract_input_hash** - id of the proposed contract

function ReleaseCodeCheckedContract

```
rpc ReleaseCodeCheckedContract (ReleaseContractInput) returns (google.protobuf.Empty) {}

message ReleaseContractInput {
    aelf.Hash proposal_id = 1;
    aelf.Hash proposed_contract_input_hash = 2;
}
```

Release the proposal which has passed the code check.

Parameters:

ReleaseContractInput

- **proposal_id** - hash of the proposal
- **proposed_contract_input_hash** - id of the proposed contract

function DeploySmartContract

```
rpc DeploySmartContract (ContractDeploymentInput) returns (aelf.Address) {}

message ContractDeploymentInput {
    sint32 category = 1;
    bytes code = 2;
}
```

Deploys a smart contract on chain.

Parameters:

ContractDeploymentInput

- **category** - contract type (usually 0)
- **code** - byte array of the contract code

Returns:

Address of the deployed smart contract.

function UpdateSmartContract

```
rpc UpdateSmartContract (ContractUpdateInput) returns (aelf.Address) {}

message ContractUpdateInput {
    aelf.Address address = 1;
    bytes code = 2;
}
```

Updates a smart contract on chain.

Parameters:

ContractUpdateInput

- **address** - address of the smart contract to be updated
- **code** - byte array of the updated contract code

Returns:

Address of the updated smart contract.

function Initialize

```
rpc Initialize (InitializeInput) returns (google.protobuf.Empty) {}

message InitializeInput{
    bool contract_deployment_authority_required = 1;
}
```

Initializes the genesis contract.

Parameters:

InitializeInput

- **contract_deployment_authority_required** - whether contract deployment/update requires authority

function ChangeGenesisOwner

```
rpc ChangeGenesisOwner (aelf.Address) returns (google.protobuf.Empty) {}
```

Change the owner of the genesis contract.

Parameters:

- **Address** - address of new genesis owner

function SetContractProposerRequiredState

```
rpc SetContractProposerRequiredState (google.protobuf.BoolValue) returns (google.
↪protobuf.Empty) {}
```

Set authority of contract deployment.

Parameters:

- **google.protobuf.BoolValue** - whether contract deployment/update requires contract proposer authority

function ChangeContractDeploymentController

```
rpc ChangeContractDeploymentController (acs1.AuthorityInfo) returns (google.protobuf.
↪Empty) {}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Modify the contract deployment controller authority. Note: Only old controller has permission to do this.

Parameters:

- **AuthorityInfo** - new controller authority info containing organization address and contract address that the organization belongings to

function ChangeCodeCheckController

```
rpc ChangeCodeCheckController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Modifies the contract code check controller authority. Note: Only old controller has permission to do this.

Parameters:

- **AuthorityInfo** - new controller authority info containing organization address and contract address that the organization belongings to

function SetInitialControllerAddress

```
rpc SetInitialControllerAddress (aelf.Address) returns (google.protobuf.Empty) {}
```

Sets initial controller address for **CodeCheckController** and **ContractDeploymentController**

Parameters:

- **Address** - initial controller (which should be parliament organization as default)

function GetContractDeploymentController

```
rpc GetContractDeploymentController (google.protobuf.Empty) returns (acs1.AuthorityInfo)
↪{
        option (aelf.is_view) = true;
}
message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Returns **ContractDeploymentController** authority info.

Returns:

- **AuthorityInfo** - **ContractDeploymentController** authority info.

function GetContractDeploymentController

```
rpc GetCodeCheckController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {
        option (aelf.is_view) = true;
}
message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Returns **CodeCheckController** authority info.

Returns:

- **AuthorityInfo** - **CodeCheckController** authority info.

2.7 multi-token

The multi-token contract is most essentially used for managing balances.

2.7.1 Token life-cycle: creation, issuance and transfer.

These methods constitute the basic functionality needed to maintain balances for tokens. For a full listing of the contracts methods you can check the Token Contract definition on GitHub.

Create

```
rpc Create (CreateInput) returns (google.protobuf.Empty) { }

message CreateInput {
    string symbol = 1;
    string token_name = 2;
    sint64 total_supply = 3;
    sint32 decimals = 4;
    aelf.Address issuer = 5;
    bool is_burnable = 6;
    repeated aelf.Address lock_white_list = 7;
    bool is_profitable = 8;
    int32 issue_chain_id = 9;
}
```

The token contract permits the creation of an entirely new token and the first action needed before using a token is its creation. The **Create** method takes exactly on parameter, a **CreateInput** message.

- **issuer** is the creator of this token.
- **symbol** is a short string between 1 and 8 characters composed only of upper-case letters like for example "ELF" or "AETC" (no numbers allowed). Of course, since tokens are uniquely identified by the symbol, it must not already exist.
- **token_name** is a more descriptive name for your token or the long name. For example, "RMB" could be the token symbol and "RenMinBi" the token's name. This is a non-optional field up to 80 characters in length.
- **total_supply** for the token is the amount of tokens that will exist. This must be larger than 0.
- **decimals** is a positive integer between 0-18.
- **issue_chain_id** is the id of the chain, this defaults to the chain id of the node.

Issue

```
rpc Issue (IssueInput) returns (google.protobuf.Empty) { }

message IssueInput {
    string symbol = 1;
    sint64 amount = 2;
    string memo = 3;
    aelf.Address to = 4;
}
```

Issuing some amount of tokens to an address is the action of increasing that addresses balance for the given token. The total amount of issued tokens must not exceed the total supply of the token and only the issuer (creator) of the token can issue tokens. Issuing tokens effectively increases the circulating supply. The **Issue** method takes exactly one parameter, a **IssueInput** message.

- **symbol** is the symbol that identifies the token, it must exist.
- **amount** is the amount to issue.
- **to** field the receiver address of the newly issued tokens.
- **memo** optionally you can specify a later accessible when parsing the transaction.

Transfer

```
rpc Transfer (TransferInput) returns (google.protobuf.Empty) { }

message TransferInput {
    aelf.Address to = 1;
    string symbol = 2;
    sint64 amount = 3;
    string memo = 4;
}
```

Transferring tokens simply is the action of transferring a given amount of tokens from one address to another. The origin or source address is the signer of the transaction. The balance of the sender must be higher than the amount that is transferred. The **Transfer** method takes exactly one parameter, a **TransferInput** message.

- **to** field is the receiver of the tokens.
- **symbol** is the symbol that identifies the token, it must exist.
- **amount** is the amount to to transfer.
- **memo** optionally you can specify a later accessible when parsing the transaction.

TransferFrom

```
rpc TransferFrom (TransferFromInput) returns (google.protobuf.Empty) { }

message TransferFromInput {
    aelf.Address from = 1;
    aelf.Address to = 2;
    string symbol = 3;
    sint64 amount = 4;
    string memo = 5;
}
```

The **TransferFrom** action will transfer a specified amount of tokens from one address to another. For this operation to succeed the **from** address needs to have approved (see *allowances*) enough tokens to Sender of this transaction. If successful the amount will be removed from the allowance.

- **from** the source address of the tokens.
- **to** the destination address of the tokens.
- **symbol** the symbol of the token to transfer.
- **amount** the amount to transfer.
- **memo** an optional memo.

2.7.2 Allowances

Allowances allow some entity (in fact an address) to authorize another address to transfer tokens on his behalf (see **TransferFrom**). There are two methods available for controlling this, namely **Approve** and **UnApprove**.

Approve

```
rpc Approve (ApproveInput) returns (google.protobuf.Empty) { }

message ApproveInput {
    aelf.Address spender = 1;
    string symbol = 2;
    sint64 amount = 3;
}
```

The approve action increases the allowance from the *Sender* to the **Spender** address, enabling the Spender to call **TransferFrom**.

- **spender** the address that will have it's allowance increased.
- **symbol** the symbol of the token to approve.
- **amount** the amount of tokens to approve.

UnApprove

```
rpc UnApprove (UnApproveInput) returns (google.protobuf.Empty) { }

message UnApproveInput {
    aelf.Address spender = 1;
    string symbol = 2;
    sint64 amount = 3;
}
```

This is the reverse operation for **Approve**, it will decrease the allowance.

- **spender** the address that will have it's allowance decreased.
- **symbol** the symbol of the token to un-approve.
- **amount** the amount of tokens to un-approve.

2.7.3 Locking

Lock

```
rpc Lock (LockInput) returns (google.protobuf.Empty) { }

message LockInput {
    aelf.Address address = 1;
    aelf.Hash lock_id = 2;
    string symbol = 3;
```

(continues on next page)

(continued from previous page)

```
    string usage = 4;
    int64 amount = 5;
}
```

This method can be used to lock tokens.

- **address** the entity that wants to lock its tokens.
- **lock_id** id of the lock.
- **symbol** the symbol of the token to lock.
- **usage** a memo.
- **amount** the amount of tokens to lock.

Unlock

```
rpc Unlock (UnlockInput) returns (google.protobuf.Empty) { }

message UnlockInput {
    aelf.Address address = 1; // The one want to lock his token.
    aelf.Hash lock_id = 2;
    string symbol = 3;
    string usage = 4;
    int64 amount = 5;
}
```

This is the reverse operation of locking, it un-locks some previously locked tokens.

- **address** the entity that wants to un-lock its tokens.
- **lock_id** id of the lock.
- **symbol** the symbol of the token to un-lock.
- **usage** a memo.
- **amount** the amount of tokens to un-lock.

2.7.4 Burning tokens

Burn

```
rpc Burn (BurnInput) returns (google.protobuf.Empty) { }

message BurnInput {
    string symbol = 1;
    sint64 amount = 2;
}
```

This action will burn the specified amount of tokens, removing them from the token's *Supply*

- **symbol** the symbol of the token to burn.
- **amount** the amount of tokens to burn.

2.7.5 Cross-chain

CrossChainCreateToken

```
rpc CrossChainCreateToken(CrossChainCreateTokenInput) returns (google.protobuf.Empty) { }

message CrossChainCreateTokenInput {
    int32 from_chain_id = 1;
    int64 parent_chain_height = 2;
    bytes transaction_bytes = 3;
    aelf.MerklePath merkle_path = 4;
}
```

This action is used for creating a "cross-chain" token. This action should be called on the side-chain's with the information about the transaction that created the token on the parent chain.

- **from_chain_id** the chain id of the chain on which the token was created.
- **parent_chain_height** the height of the transaction that created the token.
- **transaction_bytes** the transaction that created the token.
- **merkle_path** the merkle path created from the transaction that created the transaction.

CrossChainTransfer

```
rpc CrossChainTransfer (CrossChainTransferInput) returns (google.protobuf.Empty) { }

message CrossChainTransferInput {
    aelf.Address to = 1;
    string symbol = 2;
    sint64 amount = 3;
    string memo = 4;
    int32 to_chain_id = 5;
    int32 issue_chain_id = 6;
}
```

This action is used for transferring tokens across chains, this effectively burn the tokens on the chain.

- **to** the receiving account.
- **symbol** the token.
- **amount** the amount of tokens that will be transferred.
- **memo** an optional memo.
- **to_chain_id** the destination chain id.
- **issue_chain_id** the source chain id.

CrossChainReceiveToken

```
rpc CrossChainReceiveToken (CrossChainReceiveTokenInput) returns (google.protobuf.Empty)
↪{ }
```

(continues on next page)

(continued from previous page)

```
message CrossChainReceiveTokenInput {
    int32 from_chain_id = 1;
    int64 parent_chain_height = 2;
    bytes transfer_transaction_bytes = 3;
    aelf.MerklePath merkle_path = 4;
}
```

This method is used on the destination chain for receiving tokens after a **Transfer** operation.

- **from_chain_id** the source chain.
- **parent_chain_height** the height of the transfer transaction.
- **transfer_transaction_bytes** the raw bytes of the transfer transaction.
- **merkle_path** the merkle path created from the transfer transaction.

SetSymbolsToPayTxSizeFee

```
rpc SetSymbolsToPayTxSizeFee (SymbolListToPayTxSizeFee) returns (google.protobuf.Empty){
}

message SymbolListToPayTxSizeFee{
    repeated SymbolToPayTxSizeFee symbols_to_pay_tx_size_fee = 1;
}

message SymbolToPayTxSizeFee{
    string token_symbol = 1;
    sint32 base_token_weight = 2;
    sint32 added_token_weight = 3;
}
```

This action sets available tokens that can be used to pay for transaction fee.

- **symbols_to_pay_tx_size_fee** available token list.
  - **token_symbol** token symbol.
  - **base_token_weight** it is fixed to primary token.
  - **added_token_weight** if base_token_weight set to 1 and added_token_weight set to 10, it will cost 10 this token instead of primary token.

UpdateCoefficientsForContract

```
message UpdateCoefficientsInput {
    repeated sint32 piece_numbers = 1;// To specify pieces gonna update.
    CalculateFeeCoefficients coefficients = 2;
}

message CalculateFeeCoefficients {
    sint32 fee_token_type = 1;
    repeated CalculateFeePieceCoefficients piece_coefficients_list = 2;
}
```

(continues on next page)

(continued from previous page)

```
message CalculateFeePieceCoefficients {
    repeated sint32 value = 1;
}

enum FeeTypeEnum {
    READ = 0;
    STORAGE = 1;
    WRITE = 2;
    TRAFFIC = 3;
    TX = 4;
}
```

This action sets methods used to calculate resource token fees.

- **fee_token_type** resource fee type (exclude TX).
- **piece_coefficients_list** it is a coefficients array.
  - **value** it is a int array. its first element indicates its piece key. other every three consecutive elements indicates a function, like (2, 1, 1) means (1/1) * x^2.

UpdateCoefficientsForSender

```
rpc UpdateCoefficientsForSender (UpdateCoefficientsInput) returns (google.protobuf.
↪Empty) {
}

message UpdateCoefficientsInput {
    repeated sint32 piece_numbers = 1;// To specify pieces gonna update.
    CalculateFeeCoefficients coefficients = 2;
}

message CalculateFeePieceCoefficients {
    repeated sint32 value = 1;
}
```

This action sets methods used to calculate transaction fee.

note: *for CalculateFeeCoefficients see UpdateCoefficientsForContract*

AdvanceResourceToken

```
rpc AdvanceResourceToken (AdvanceResourceTokenInput) returns (google.protobuf.Empty) {
}

message AdvanceResourceTokenInput {
    aelf.Address contract_address = 1;
    string resource_token_symbol = 2;
    sint64 amount = 3;
}
```

This action transfers resource tokens to designated contract address.

- **contract_address** the contract address.
- **resource_token_symbol** resource token symbol.
- **amount** the amount of tokens.

TakeResourceTokenBack

```
rpc TakeResourceTokenBack (TakeResourceTokenBackInput) returns (google.protobuf.Empty) {
}

message TakeResourceTokenBackInput {
    aelf.Address contract_address = 1;
    string resource_token_symbol = 2;
    sint64 amount = 3;
}
```

This method takes token from contract address

- **contract_address** the contract address.
- **resource_token_symbol** resource token symbol.
- **amount** the amount of tokens.

ValidateTokenInfoExists

```
rpc ValidateTokenInfoExists(ValidateTokenInfoExistsInput) returns (google.protobuf.Empty)
↪{
}

message ValidateTokenInfoExistsInput{
    string symbol = 1;
    string token_name = 2;
    sint64 total_supply = 3;
    sint32 decimals = 4;
    aelf.Address issuer = 5;
    bool is_burnable = 6;
    sint32 issue_chain_id = 7;
    bool is_profitable = 8;
}
```

This method validates if the token exist.

- **symbol** the token symbol.
- **token_name** the token name.
- **total_supply** total supply of the token.
- **decimals** decimals.
- **issuer** the token issuer.
- **is_burnable** indicates if the token is burnable.
- **issue_chain_id** issue chain id.

- **is_profitable** indicates if the token is profitable.

TransferToContract

```
rpc TransferToContract (TransferToContractInput) returns (google.protobuf.Empty) {
}

message TransferToContractInput {
    string symbol = 1;
    sint64 amount = 2;
    string memo = 3;
}
```

This method transfer token to token address.

- **symbol** the token symbol.
- **amount** amount.
- **memo** transfer memo.

InitializeAuthorizedController

```
rpc InitializeAuthorizedController(google.protobuf.Empty) returns (google.protobuf.Empty)
↪{
}
```

This method initializes the controller for calling UpdateCoefficientsForContract and UpdateCoefficientsForSender. Note that, if the current chain is side chain, it will create a controller for managing chain rental.

ChangeUserFeeController

```
rpc ChangeUserFeeController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: controller type.
- **owner address**: controller's address.

This method change the controller who sets the coefficient for calculating transaction size fee.

ChangeDeveloperController

```
rpc ChangeDeveloperController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {}

message AuthorityInfo {
```

(continues on next page)

(continued from previous page)

```
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: controller type.
- **owner address**: controller's address.

This method change the controller who sets the coefficient for calculating resource token.

2.7.6 View methods

GetTokenInfo

```
rpc GetTokenInfo (GetTokenInfoInput) returns (TokenInfo) { }

message GetTokenInfoInput {
    string symbol = 1;
}

message TokenInfo {
    string symbol = 1;
    string token_name = 2;
    sint64 supply = 3;
    sint64 total_supply = 4;
    sint32 decimals = 5;
    aelf.Address issuer = 6;
    bool is_burnable = 7;
    bool is_profitable = 8;
    sint32 issue_chain_id = 9;
    sint64 burned = 10;
}
```

This view method returns a **TokenInfo** object that describes information about a token.

Input:

- **symbol** the token for which you want the information.

Output:

- **symbol** the symbol of the token.
- **token_name** the full name of the token.
- **supply** the current supply of the token.
- **total_supply** the total supply of the token.
- **decimals** the amount of decimal places this token has.
- **issuer** the address that created the token.
- **is_burnable** a flag indicating if this token is burnable.
- **is_profitable** a flag indicating if this token is profitable.

- **issue_chain_id** the chain of this token.
- **burned** the amount of burned tokens.

GetNativeTokenInfo

```
rpc GetNativeTokenInfo (google.protobuf.Empty) returns (TokenInfo) { }
```

note: *for TokenInfo see GetTokenInfo*

This view method returns the TokenInfo object associated with the native token.

GetResourceTokenInfo

```
rpc GetResourceTokenInfo (google.protobuf.Empty) returns (TokenInfoList) { }

message TokenInfoList {
    repeated TokenInfo value = 1;
}
```

note: *for TokenInfo see GetTokenInfo*

This view method returns the list of TokenInfo objects associated with the chain's resource tokens.

GetBalance

```
rpc GetBalance (GetBalanceInput) returns (GetBalanceOutput) { }

message GetBalanceInput {
    string symbol = 1;
    aelf.Address owner = 2;
}

message GetBalanceOutput {
    string symbol = 1;
    aelf.Address owner = 2;
    sint64 balance = 3;
}
```

This view method returns the balance of an address.

Input:

- **symbol** the token for which to get the balance.
- **owner** the address for which to get the balance.

Output:

- **symbol** the token for which to get the balance.
- **owner** the address for which to get the balance.
- **balance** the current balance.

GetAllowance

```
rpc GetAllowance (GetAllowanceInput) returns (GetAllowanceOutput) { }

message GetAllowanceInput {
    string symbol = 1;
    aelf.Address owner = 2;
    aelf.Address spender = 3;
}

message GetAllowanceOutput {
    string symbol = 1;
    aelf.Address owner = 2;
    aelf.Address spender = 3;
    sint64 allowance = 4;
}
```

This view method returns the allowance of one address to another.

Input:

- **symbol** the token for which to get the allowance.
- **owner** the address for which to get the allowance (that approved tokens).
- **spender** the address of the spender.

Output:

- **symbol** the token for which to get the allowance.
- **owner** the address for which to get the allowance (that approved tokens).
- **spender** the address of the spender.
- **allowance** the current allowance.

IsInWhiteList

```
rpc IsInWhiteList (IsInWhiteListInput) returns (google.protobuf.BoolValue) { }

message IsInWhiteListInput {
    string symbol = 1;
    aelf.Address address = 2;
}
```

This method returns wether or not the given address is in the lock whitelist.

- **symbol** the token.
- **address** the address that is checked.

GetLockedAmount

```
rpc GetLockedAmount (GetLockedAmountInput) returns (GetLockedAmountOutput) { }

message GetLockedAmountInput {
    aelf.Address address = 1;
    string symbol = 2;
    aelf.Hash lock_id = 3;
}

message GetLockedAmountOutput {
    aelf.Address address = 1;
    string symbol = 2;
    aelf.Hash lock_id = 3;
    sint64 amount = 4;
}
```

This view method returns the amount of tokens currently locked by an address.

Input:

- **address** the address.
- **symbol** the token.
- **lock_id** the lock id.

Output:

- **address** the address.
- **symbol** the token.
- **lock_id** the lock id.
- **amount** the amount currently locked by the specified address.

GetCrossChainTransferTokenContractAddress

```
rpc GetCrossChainTransferTokenContractAddress␣
↪(GetCrossChainTransferTokenContractAddressInput) returns (aelf.Address) { }

message GetCrossChainTransferTokenContractAddressInput {
    int32 chainId = 1;
}
```

This view method returns the cross-chain transfer address for the given chain.

- **chainId** the id of the chain.

GetPrimaryTokenSymbol

```
rpc GetPrimaryTokenSymbol (google.protobuf.Empty) returns (google.protobuf.StringValue) {
```

This view method return the primary token symbol if it's set. If not, returns the Native symbol.

GetCalculateFeeCoefficientOfContract

```
rpc GetCalculateFeeCoefficientForContract (aelf.SInt32Value) returns␣
↪(CalculateFeeCoefficients) { }

message CalculateFeeCoefficients {
    sint32 fee_token_type = 1;
    repeated CalculateFeePieceCoefficients piece_coefficients_list = 2;
}

message CalculateFeePieceCoefficients {
    repeated sint32 value = 1;
}

enum FeeTypeEnum {
    READ = 0;
    STORAGE = 1;
    WRITE = 2;
    TRAFFIC = 3;
    TX = 4;
}
```

This view method returns the resource tokens fee calculation method.

Input resource fee type.

Output note: *for CalculateFeeCoefficients see UpdateCoefficientsForContract*

GetCalculateFeeCoefficientOfSender

```
rpc GetCalculateFeeCoefficientForSender (google.protobuf.Empty) returns␣
↪(CalculateFeeCoefficients) { }
```

This view method returns transaction fee's calculation method.

note: *for CalculateFeeCoefficients see GetCalculateFeeCoefficientForContract*

GetSymbolsToPayTxSizeFee

```
rpc GetSymbolsToPayTxSizeFee (google.protobuf.Empty) returns (SymbolListToPayTxSizeFee){
    option (aelf.is_view) = true;
}

message SymbolListToPayTxSizeFee{
    repeated SymbolToPayTxSizeFee symbols_to_pay_tx_size_fee = 1;
}

message SymbolToPayTxSizeFee{
    string token_symbol = 1;
    sint32 base_token_weight = 2;
    sint32 added_token_weight = 3;
}
```

This method returns available tokens that can be used to pay for transaction fee.

note: *for SymbolListToPayTxSizeFee see SetSymbolsToPayTxSizeFee*

GetDeveloperFeeController

```
rpc GetDeveloperFeeController (google.protobuf.Empty) returns (DeveloperFeeController) {
}

message DeveloperFeeController {
    acs1.AuthorityInfo root_controller = 1;
    acs1.AuthorityInfo parliament_controller = 2;
    acs1.AuthorityInfo developer_controller = 3;
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

This method returns the controller for UpdateCoefficientsForContract. The root address consists originally of default parliament organization, developer organization. The type of root controller and developer controller is Assocation.

- **root_controller** root controller information.
- **parliament_controller** parliament controller information.
- **developer_controller** developer controller information.
- **contract_address** in which contract the organization is created.
- **owner_address** organization address

GetUserFeeController

```
rpc GetUserFeeController (google.protobuf.Empty) returns (UserFeeController) {
}

message UserFeeController{
    acs1.AuthorityInfo root_controller = 1;
    acs1.AuthorityInfo parliament_controller = 2;
    acs1.AuthorityInfo referendum_controller = 3;
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

This method returns the controller for UpdateCoefficientsForSender. The root address consists originally of default parliament organization, referendum organization. The type of root controller and developer controller is Assocation.

- **root_controller** root controller information.

- **parliament_controller** parliament controller information.
- **referendum_controller** referndum controller information.
- **contract_address** in which contract the organization is created.
- **owner_address** organization address

GetSideChainRentalControllerCreateInfo

```
rpc GetSideChainRentalControllerCreateInfo (google.protobuf.Empty) returns (acs1.
↪AuthorityInfo) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: controller type.
- **owner address**: controller's address.

GetResourceUsage

```
rpc GetResourceUsage (google.protobuf.Empty) returns (ResourceUsage) {
}

message ResourceUsage {
    map<string, sint32> value = 1;
}
```

This method is used on a side chain. It returns how much resource tokens should be paid at the moment.

- **value** resource token symbol => amount.

GetOwningRental

```
rpc GetOwningRental (google.protobuf.Empty) returns (OwningRental) {
}

message OwningRental {
    map<string, sint64> resource_amount = 1;
}
```

This method is used on a side chain. It returns how much resource tokens (count * value) should be paid at the moment.

- **resource_amount** resource token symbol => amount.

GetOwningRentalUnitValue

```
rpc GetOwningRentalUnitValue (google.protobuf.Empty) returns (OwningRentalUnitValue) {
}

message OwningRentalUnitValue {
    map<string, sint64> resource_unit_value = 1;
}
```

This method is used in side chain. It returns resouces token' unit value. (pay = unit value * amount)

- **resource_unit_value** resource token symbol => unit value.

OwningRentalUnitValue

```
rpc GetUserFeeController(google.protobuf.Empty) returns (UserFeeController){}

message UserFeeController{
    acs1.AuthorityInfo root_controller = 1;
    acs1.AuthorityInfo parliament_controller = 2;
    acs1.AuthorityInfo referendum_controller = 3;
}
```

Get the controllers(By defalult, the contoller consist of parliament and referendum). If you change the controller, just the root controller has value.

returns:

- **root controller**: the root controller, it is a association by default.
- **parliament controller**: parliament controller, member of the root controller.
- **referendum controller**: referendum controller, member of the root controller.

2.8 Profit Contract

The Profit contract is an abstract layer for creating scheme to share bonus. Developers can build a system to distribute bonus by call this contract.

2.8.1 Scheme Creation

This method creates a new scheme based on the **CreateSchemeInput** message.

```
rpc CreateScheme (CreateSchemeInput) returns (aelf.Hash) {}

message CreateSchemeInput {
    sint64 profit_receiving_due_period_count = 1;
    bool is_release_all_balance_every_time_by_default = 2;
    sint32 delay_distribute_period_count = 3;
    aelf.Address manager = 4;
    bool can_remove_beneficiary_directly = 5;
    aelf.Hash token = 6;
```

(continues on next page)

(continued from previous page)

```
}

message SchemeCreated {
    aelf.Address virtual_address = 1;
    aelf.Address manager = 2;
    sint64 profit_receiving_due_period_count = 3;
    bool is_release_all_balance_every_time_by_default = 4;
    aelf.Hash scheme_id = 5;
}
```

CreateSchemeInput:

- **manager**: the scheme manager's Address, defaults to the transaction sender.
- **profit receiving due period_count** optional, defaults to 10.
- **is release all balance every time by default** if true, all the schemes balance will be distributed during distribution if the input amount is 0.
- **delay distribute period count** distribute bonus after terms.
- **can remove beneficiary directly** indicates whether the beneficiary can be removed without considering its EndPeriod and IsWeightRemoved.
- **token** used to indicates scheme id.

returns:

- **value**: the newly created scheme id.

After a successful creation, a **SchemeCreated** event log can be found in the transaction result.

SchemeCreated:

- **virtual address**: transfer from scheme id.
- **manager**: manager address.
- **scheme id**: scheme id.

2.8.2 Add sub-scheme

Two previously created schemes can be put in a scheme/sub-scheme relation. This will effectively add the specified sub-scheme as a **beneficiary** of the parent scheme.

```
rpc AddSubScheme (AddSubSchemeInput) returns (google.protobuf.Empty) {}

message AddSubSchemeInput {
    aelf.Hash scheme_id = 1;
    aelf.Hash sub_scheme_id = 2;
    sint64 sub_scheme_shares = 3;
}
```

- **scheme id**: the parent scheme ID.
- **sub scheme id**: the child scheme ID.
- **sub scheme shares**: number of shares of the sub-scheme.

2.8.3 Remove sub-scheme

Removes a sub-scheme from a scheme. Note that only the manager of the parent scheme can remove a sub-scheme from it.

```
rpc RemoveSubScheme (RemoveSubSchemeInput) returns (google.protobuf.Empty) {}

message RemoveSubSchemeInput {
    aelf.Hash scheme_id = 1;
    aelf.Hash sub_scheme_id = 2;
}
```

RemoveSubSchemeInput:

- **scheme id**: scheme id
- **sub scheme id**: sub-scheme id

2.8.4 Add beneficiary

Adds a beneficiary to a scheme. This beneficiary is either a scheme or another entity that can be represented by an AElf address.

```
rpc AddBeneficiary (AddBeneficiaryInput) returns (google.protobuf.Empty) {}

message AddBeneficiaryInput {
    aelf.Hash scheme_id = 1;
    BeneficiaryShare beneficiary_share = 2;
    sint64 end_period = 3;
}

message BeneficiaryShare {
    aelf.Address beneficiary = 1;
    sint64 shares = 2;
}
```

AddBeneficiaryInput:

- **scheme id**: scheme id.
- **beneficiary share**: share information to beneficiary.
- **end period**: end time.

BeneficiaryShare:

- **beneficiary**: beneficiary address
- **shares**: shares attributed to this beneficiary.

2.8.5 Remove beneficiary

Removes a beneficiary from a scheme.

```
rpc RemoveBeneficiary (RemoveBeneficiaryInput) returns (google.protobuf.Empty) {}

message RemoveBeneficiaryInput {
    aelf.Address beneficiary = 1;
    aelf.Hash scheme_id = 2;
}
```

RemoveBeneficiaryInput:

- **beneficiary** beneficiary address to be removed
- **scheme id** scheme id

2.8.6 Add beneficiaries

Adds multiple beneficiaries to a scheme until the given end period.

```
rpc AddBeneficiaries (AddBeneficiariesInput) returns (google.protobuf.Empty) {}

message AddBeneficiariesInput {
    aelf.Hash scheme_id = 1;
    repeated BeneficiaryShare beneficiary_shares = 2;
    sint64 end_period = 4;
}

message BeneficiaryShare {
    aelf.Address beneficiary = 1;
    sint64 shares = 2;
}
```

AddBeneficiariesInput:

- **scheme id**: scheme id.
- **beneficiary shares**: share information to beneficiaries.
- **end period**: end time.

BeneficiaryShare:

- **beneficiary**: beneficiary address.
- **shares**: shares to beneficiary.

2.8.7 Remove beneficiaries

Remove beneficiaries from a scheme.

```
 rpc RemoveBeneficiaries (RemoveBeneficiariesInput) returns (google.protobuf.Empty){}

message RemoveBeneficiariesInput {
    repeated aelf.Address beneficiaries = 1;
    aelf.Hash scheme_id = 2;
}
```

RemoveBeneficiariesInput:

- **beneficiaries** beneficiaries' addresses to be removed.
- **scheme id** scheme id.

2.8.8 Profit contribution

Contribute profit to a scheme.

```
rpc ContributeProfits (ContributeProfitsInput) returns (google.protobuf.Empty) {}

message ContributeProfitsInput {
    aelf.Hash scheme_id = 1;
    sint64 amount = 2;
    sint64 period = 3;
    string symbol = 4;
}
```

ContributeProfitsInput:

- **scheme id**: scheme id.
- **amount**: amount token contributed to the scheme.
- **period**: in which term the amount is added.
- **symbol**: token symbol.

2.8.9 Claim profits

Used to claim the profits of a given symbol. The beneficiary is identified as the sender of the transaction.

```
rpc ClaimProfits (ClaimProfitsInput) returns (google.protobuf.Empty) {}

message ClaimProfitsInput {
    aelf.Hash scheme_id = 1;
    string symbol = 2;
    aelf.Address beneficiary = 3;
}
```

ContributeProfitsInput:

- **scheme id**: scheme id.
- **symbol**: token symbol.
- **beneficiary**: optional, claiming profits for another address, transaction fees apply to the caller.

2.8.10 Distribute profits

Distribute profits to scheme (address) including its sub scheme according to term and symbol, should be called by the manager.

```
rpc DistributeProfits (DistributeProfitsInput) returns (google.protobuf.Empty) {}

message DistributeProfitsInput {
```

(continues on next page)

(continued from previous page)

```
    aelf.Hash scheme_id = 1;
    sint64 period = 2;
    sint64 amount = 3;
    string symbol = 4;
}
```

DistributeProfitsInput:

- **scheme id**: scheme id.
- **period**: term here should be the current term.
- **amount**: number.
- **symbol**: token symbol.

2.8.11 Reset manager

Reset the manager of a scheme.

```
rpc ResetManager (ResetManagerInput) returns (google.protobuf.Empty) {}

message ResetManagerInput {
    aelf.Hash scheme_id = 1;
    aelf.Address new_manager = 2;
}
```

ResetManagerInput:

- **scheme id**: scheme id.
- **new manager**: new manager's address.

2.8.12 view methods

For reference, you can find here the available view methods.

GetManagingSchemeIds

Get all schemes created by the specified manager.

```
rpc GetManagingSchemeIds (GetManagingSchemeIdsInput) returns (CreatedSchemeIds) {}

message GetManagingSchemeIdsInput {
    aelf.Address manager = 1;
}

message CreatedSchemeIds {
    repeated aelf.Hash scheme_ids = 1;
}
```

GetManagingSchemeIdsInput:

- **manager**: manager's address.

returns:

- **scheme ids**: list of scheme ids.

GetScheme

Returns the scheme with the given hash (scheme ID).

```
rpc GetScheme (aelf.Hash) returns (Scheme) {}
```

Hash:

- **value**: scheme id.

SchemeBeneficiaryShare:

- **scheme id**: sub scheme's id.
- **shares**: sub scheme shares.

GetSchemeAddress

Returns the schemes virtual address if the input period is 0 or will give the distributed profit address for the given period.

```
rpc GetSchemeAddress (SchemePeriod) returns (aelf.Address) {}

message SchemePeriod {
    aelf.Hash scheme_id = 1;
    sint64 period = 2;
}
```

SchemePeriod:

- **scheme id**: scheme id.
- **period**: period number.

returns:

- **value**: scheme's virtual address.

GetDistributedProfitsInfo

Get distributed profits Info for a given period.

```
rpc GetDistributedProfitsInfo (SchemePeriod) returns (DistributedProfitsInfo) {}

message SchemePeriod {
    aelf.Hash scheme_id = 1;
    sint64 period = 2;
}

message DistributedProfitsInfo {
    sint64 total_shares = 1;
    map<string, sint64> profits_amount = 2;
```

(continues on next page)

(continued from previous page)

```
    bool is_released = 3;
}
```

SchemePeriod:

- **scheme_id**: scheme id.
- **period**: term number.

returns:

- **total shares**: total shares, -1 indicates failed to get the information.
- **profits amount**: token symbol => reside amount.
- **is released**: is released.

GetProfitDetails

Gets a beneficiaries profit details for a given scheme.

```
rpc GetProfitDetails (GetProfitDetailsInput) returns (ProfitDetails) {}

message GetProfitDetailsInput {
    aelf.Hash scheme_id = 1;
    aelf.Address beneficiary = 2;
}

message ProfitDetails {
    repeated ProfitDetail details = 1;
}

message ProfitDetail {
    sint64 start_period = 1;
    sint64 end_period = 2;
    sint64 shares = 3;
    sint64 last_profit_period = 4;
    bool is_weight_removed = 5;
}
```

GetProfitDetailsInput:

- **scheme id** scheme id.
- **beneficiary** beneficiary.

returns:

- **details** profit details.

ProfitDetail:

- **start period**: start period.
- **end period**: end period.
- **shares**: shares indicating the weight used to calculate the profit in the future.
- **last profit period**: last period the scheme distribute.

- **is weight removed**: is it expired.

GetProfitAmount

Calculate profits (have not yet received) before current term(at most 10 term).

```
rpc GetProfitAmount (ClaimProfitsInput) returns (aelf.SInt64Value) {}

message ClaimProfitsInput {
    aelf.Hash scheme_id = 1;
    string symbol = 2;
}

message SInt64Value
{
    sint64 value = 1;
}
```

ClaimProfitsInput:

- **scheme_id**: scheme id.
- **symbol**: token symbol.

returns:

- **value**: amount of tokens.

2.9 resource

The TokenConverter contract is most essentially used for managing resources.

2.9.1 Buying and selling resources:

The token converter's contract permits buying and selling resource based on the **Bancor** algorithm.

```
rpc Buy (BuyInput) returns (google.protobuf.Empty) {}
rpc Sell (SellInput) returns (google.protobuf.Empty) {}

message BuyInput {
    string symbol = 1;
    sint64 amount = 2;
    sint64 pay_limit = 3;
}

message SellInput {
    string symbol = 1;
    sint64 amount = 2;
    sint64 receive_limit = 3;
}

// Events
```

(continues on next page)

(continued from previous page)

```
message TokenBought {
    option (aelf.is_event) = true;
    string symbol = 1 [(aelf.is_indexed) = true];
    sint64 bought_amount = 2;
    sint64 base_amount = 3;
    sint64 fee_amount = 4;
}

message TokenSold {
    option (aelf.is_event) = true;
    string symbol = 1 [(aelf.is_indexed) = true];
    sint64 sold_amount = 2;
    sint64 base_amount = 3;
    sint64 fee_amount = 4;
}
```

Buying resource requires a **BuyInput** message as parameter:

- **symbol** is token symbol to buy, it effectively describes which connector will be used.
- **amount** the amount of tokens to buy.
- **pay limit** is used to cap the amount of that you are willing to pay for this exchange (0 for no limit).

After a successful buy, a **TokenBought** event log can be found in the transaction result.

On the contrary when selling resources you need to send a **SellInput** message as parameter:

- **symbol** is token symbol to sell.
- **amount** the amount of tokens to sell.
- **receive limit** is the minimum amount of tokens that you are willing to receive for this exchange (0 for no limit).

After a successful sell, a **TokenSold** event log can be found in the transaction result.

Contract management:

These next methods for managing the token converter contract are only accessible by the manager (the initial parliament organization).

```
rpc SetConnector (Connector) returns (google.protobuf.Empty) {}

message Connector {
    string symbol = 1;
    sint64 virtual_balance = 2;
    string weight = 3;
    bool is_virtual_balance_enabled = 4;
    bool is_purchase_enabled = 5;
}
```

This method will add a connector for the given symbol.

- **symbol** the target symbol of the connector.
- **weight** is a decimal that has to be a decimal between 0 and 1.
- **virtual_balance** and **is_virtual_balance_enabled** control what balance is used for the buy and sell operations.

```
rpc SetFeeRate (google.protobuf.StringValue) returns (google.protobuf.Empty) {}
```

Sets the fee rate, a string formatted decimal between 0 and 1.

```
rpc SetManagerAddress (aelf.Address) returns (google.protobuf.Empty) {}
```

Change the manager of the token converter contract.

2.9.2 view methods

For reference, you can find here the available view methods.

```
rpc GetFeeReceiverAddress (google.protobuf.Empty) returns (aelf.Address) {}
rpc GetManagerAddress (google.protobuf.Empty) returns (aelf.Address) {}
rpc GetConnector (TokenSymbol) returns (Connector) {}
rpc GetFeeRate (google.protobuf.Empty) returns (google.protobuf.StringValue) {}
rpc GetBaseTokenSymbol (google.protobuf.Empty) returns (TokenSymbol) {}
```

2.10 Cross Chain Contract

2.10.1 Functions

Detailed Description

Defines C# API functions for cross chain contract.

2.10.2 Functions Documentation

function ProposeCrossChainIndexing

```
rpc ProposeCrossChainIndexing(CrossChainBlockData) returns (google.protobuf.Empty) {}

message CrossChainBlockData {
    repeated SideChainBlockData side_chain_block_data_list = 1;
    repeated ParentChainBlockData parent_chain_block_data_list = 2;
    int64 previous_block_height = 3;
}

message SideChainBlockData {
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_status_merkle_tree_root = 3;
    int32 chain_id = 4;
}

message ParentChainBlockData {
    int64 height = 1;
    CrossChainExtraData cross_chain_extra_data = 2;
```

(continues on next page)

(continued from previous page)

```
    int32 chain_id = 3;
    aelf.Hash transaction_status_merkle_tree_root = 4;
    map<int64, aelf.MerklePath> indexed_merkle_path = 5;
    map<string, bytes> extra_data = 6;
}

message CrossChainExtraData {
    aelf.Hash transaction_status_merkle_tree_root = 1;
}

message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

Propose once cross chain indexing.

Parameters:

CrossChainBlockData

- **SideChainBlockData**
  - height : height of side chain block
  - block_header_hash : hash of side chain block
  - transaction_merkle_tree_root : merkle tree root computing from transactions status in side chain block
  - chain_id : id of side chain
- **ParentChainBlockData**
  - height : height of parent chain
  - **cross_chain_extra_data**
    - transaction_status_merkle_tree_root : the merkle tree root computing from side chain roots.
  - chain_id : parent chain id
  - transaction_status_merkle_root : merkle tree root computing from transactions status in parent chain block
  - indexed_merkle_path : <block height, merkle path> key-value map
  - extra_data : extra data map
- **previous_block_height** previous block height

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

function GetPendingCrossChainIndexingProposal

```
rpc GetPendingCrossChainIndexingProposal (google.protobuf.Empty) returns␣
↪(GetPendingCrossChainIndexingProposalOutput) {
        option (aelf.is_view) = true;
```

(continues on next page)

(continued from previous page)

```
}

message GetPendingCrossChainIndexingProposalOutput{
    aelf.Hash proposal_id = 1;
    aelf.Address proposer = 2;
    bool to_be_released = 3;
    acs7.CrossChainBlockData proposed_cross_chain_block_data = 4;
    google.protobuf.Timestamp expired_time = 5;
}

message CrossChainBlockData {
    repeated SideChainBlockData side_chain_block_data_list = 1;
    repeated ParentChainBlockData parent_chain_block_data_list = 2;
    int64 previous_block_height = 3;
}

message SideChainBlockData {
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_status_merkle_tree_root = 3;
    int32 chain_id = 4;
}

message ParentChainBlockData {
    int64 height = 1;
    CrossChainExtraData cross_chain_extra_data = 2;
    int32 chain_id = 3;
    aelf.Hash transaction_status_merkle_tree_root = 4;
    map<int64, aelf.MerklePath> indexed_merkle_path = 5;
    map<string, bytes> extra_data = 6;
}

message CrossChainExtraData {
    aelf.Hash transaction_status_merkle_tree_root = 1;
}
```

Get pending cross chain indexing proposal info.

Returns:

GetPendingCrossChainIndexingProposalOutput

- **proposal_id** - cross chain indexing proposal id
- **proposer** - proposer of cross chain indexing proposal
- **to_be_released** - true if the proposal can be released, otherwise false
- **proposed_cross_chain_block_data** - cross chain data proposed
- **expired_time** - proposal expiration time

function ReleaseCrossChainIndexing

```
rpc ReleaseCrossChainIndexing(aelf.Hash) returns (google.protobuf.Empty) {}
```

Release cross chain indexing proposal and side chain will be created.

Parameters:

Hash Cross chain indexing proposal id.

function Initialize

```
rpc Initialize (InitializeInput) returns (google.protobuf.Empty) {}
message InitializeInput
{
    int32 parent_chain_id = 1;
    int64 creation_height_on_parent_chain = 2;
    bool is_privilege_preserved = 3;
}
```

Initialize cross-chain-contract.

Parameters:

InitializeInput

- **parent_chain_id** - id of parent chain
- **creation_height_on_parent_chain** - height of side chain creation on parent chain
- **is_privilege_preserved** - true if chain privilege needed, otherwise false

function RequestSideChainCreation

```
rpc RequestSideChainCreation(SideChainCreationRequest) returns (google.protobuf.Empty){}

message SideChainCreationRequest {
    int64 indexing_price = 1;
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    string side_chain_token_symbol = 4;
    string side_chain_token_name = 5;
    int64 side_chain_token_total_supply = 6;
    int32 side_chain_token_decimals = 7;
    bool is_side_chain_token_burnable = 8;
    bool is_side_chain_token_profitable = 9;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 10;
    map<string, int32> initial_resource_amount = 11;
}

message SideChainTokenInitialIssue{
    aelf.Address address = 1;
    int64 amount = 2;
}
```

(continues on next page)

(continued from previous page)

```
message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

Request side chain creation.

Parameters:

- **SideChainCreationRequest**
  - **indexing_price** - indexing fee.
  - **locked_token_amount** - initial locked balance for a new side chain.
  - **is_privilege_preserved** - creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.
  - **side_chain_token_symbol** - side chain token symbol.
  - **side_chain_token_name** - side chain token name.
  - **side_chain_token_total_supply** - total supply of side chain token.
  - **side_chain_token_decimals** - sÏide chain token decimal.
  - **is_side_chain_token_burnable** - side chain token burnable flag.
  - **is_side_chain_token_profitable** - a flag to indicate wether the chain is profitable or not.
  - **side_chain_token_initial_issue_list** - a list of accounts and amounts that will be issued when the chain starts.
  - **initial_resource_amount** - the initial rent resources.

After a successful execution, a **ProposalCreated** event log can be found in the transaction result.

function ReleaseSideChainCreation

```
rpc ReleaseSideChainCreation(ReleaseSideChainCreationInput) returns (google.protobuf.
↪Empty){}

message ReleaseSideChainCreationInput {
    aelf.Hash proposal_id = 1;
}
```

Release side chain creation ant side chain creation proposal will be created.

Parameters:

- **ReleaseSideChainCreationInput**
  - **proposal_id** - side chain creation proposal id

function CreateSideChain

```
rpc CreateSideChain (CreateSideChainInput) returns (google.protobuf.Int32Value) {}

message CreateSideChainInput{
    SideChainCreationRequest side_chain_creation_request = 1;
    aelf.Address proposer = 2;
}

message SideChainCreationRequest {
    int64 indexing_price = 1;
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    string side_chain_token_symbol = 4;
    string side_chain_token_name = 5;
    int64 side_chain_token_total_supply = 6;
    int32 side_chain_token_decimals = 7;
    bool is_side_chain_token_burnable = 8;
    bool is_side_chain_token_profitable = 9;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 10;
    map<string, int32> initial_resource_amount = 11;
}

message SideChainTokenInitialIssue {
    aelf.Address address = 1;
    int64 amount = 2;
}
```

Create a new side chain, this is be triggered by an organization address.

Parameters:

- **CreateSideChainInput**
  - **proposer** the proposer of the proposal that triggered this method.
  - ***SideChainCreationRequest***
    - **indexing_price** - indexing fee.
    - **locked_token_amount** - initial locked balance for a new side chain.
    - **is_privilege_preserved** - creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.
    - **side_chain_token_symbol** - side chain token symbol.
    - **side_chain_token_name** - side chain token name.
    - **side_chain_token_total_supply** - total supply of side chain token.
    - **side_chain_token_decimals** - sÏide chain token decimal.
    - **is_side_chain_token_burnable** - side chain token burnable flag.
    - **is_side_chain_token_profitable** - a flag to indicate wether the chain is profitable or not.
    - **side_chain_token_initial_issue_list** - a list of accounts and amounts that will be issued when the chain starts.
    - **initial_resource_amount** - the initial rent resources.

Returns:

Id of a new side chain

function SetInitialSideChainLifetimeControllerAddress

```
rpc SetInitialSideChainLifetimeControllerAddress(aelf.Address) returns (google.protobuf.
↪Empty){}
```

Sets the initial **SideChainLifetimeController** address which should be parliament organization by default.

Parameters:

- **address** : the owner's address.

function SetInitialIndexingControllerAddress

```
rpc SetInitialIndexingControllerAddress(aelf.Address) returns (google.protobuf.Empty){}
```

Sets the initial **CrossChainIndexingController** address which should be parliament organization by default.

Parameters:

- **address** : the owner's address.

function ChangeCrossChainIndexingController

```
rpc ChangeCrossChainIndexingController(acs1.AuthorityInfo) returns (google.protobuf.
↪Empty) { }

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Changes the cross chain indexing controller.

Parameters:

- **acs1.AuthorityInfo** :
  - **contract_address** - the address of the contract that generated the controller.
  - **owner_address** - the address of the controller.

function GetCrossChainIndexingController

```
rpc GetCrossChainIndexingController(google.protobuf.Empty) returns (acs1.AuthorityInfo){
        option (aelf.is_view) = true;
}

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Get indexing fee adjustment controller for specific side chain.

Returns:

- **acs1.AuthorityInfo** :
  - **contract_address** - the address of the contract that generated the controller.
  - **owner_address** - the address of the controller.

function ChangeSideChainLifetimeController

```
rpc ChangeSideChainLifetimeController(acs1.AuthorityInfo) returns (google.protobuf.
↪Empty) { }

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Changes the side chain's lifetime controller.

Parameters:

- **acs1.AuthorityInfo** :
  - **contract_address** - the address of the contract that generated the controller.
  - **owner_address** - the address of the controller.

function GetSideChainLifetimeController

```
rpc GetSideChainLifetimeController(google.protobuf.Empty) returns (acs1.AuthorityInfo){
        option (aelf.is_view) = true;
}

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Get the side chain's lifetime controller.

Returns:

- **acs1.AuthorityInfo** :
  - **contract_address** - the address of the contract that generated the controller.
  - **owner_address** - the address of the controller.

function GetSideChainIndexingFeeController

```
rpc GetSideChainIndexingFeeController(google.protobuf.Int32Value) returns (acs1.
↪AuthorityInfo){
        option (aelf.is_view) = true;
```

(continues on next page)

(continued from previous page)

```
}

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Get side chain indexing fee.

Parameters:

- **Int32Value** : side chain id

Returns:

- **acs1.AuthorityInfo** :
  - **contract_address** - the address of the contract that generated the controller.
  - **owner_address** - the address of the controller.

function ChangeSideChainIndexingFeeController

```
rpc ChangeSideChainIndexingFeeController(ChangeSideChainIndexingFeeControllerInput)␣
↪returns (google.protobuf.Empty){}

message ChangeSideChainIndexingFeeControllerInput{
    int32 chain_id = 1;
    acs1.AuthorityInfo authority_info = 2;
}

message acs1.AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

Changes indexing fee adjustment controller for specific side chain.

Parameters:

- **ChangeSideChainIndexingFeeControllerInput** :
  - **chain_id** - side chain id.
  - **authority_info** :
    - **contract_address** - the address of the contract that generated the controller.
    - **owner_address** - the address of the controller.

function GetSideChainIndexingFeePrice

```
rpc GetSideChainIndexingFeePrice(google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value) {
        option (aelf.is_view) = true;
}
```

Get side chain indexing fee.

Parameters:

- **Int32Value** : side chain id

Returns:

- **Int64Value** : indexing fee price

function Recharge

```
rpc Recharge (RechargeInput) returns (google.protobuf.Empty) {}
message RechargeInput {
    int32 chain_id = 1;
    int64 amount = 2;
}
```

Recharge for specified side chain.

Parameters:

RechargeInput

- **chain_id** - id of the side chain
- **amount** - the token amount to recharge

function RecordCrossChainData

```
rpc RecordCrossChainData (RecordCrossChainDataInput) returns (google.protobuf.Empty) {}

message RecordCrossChainDataInput{
    CrossChainBlockData proposed_cross_chain_data = 1;
    aelf.Address proposer = 2;
}

message CrossChainBlockData {
    repeated SideChainBlockData side_chain_block_data_list = 1;
    repeated ParentChainBlockData parent_chain_block_data_list = 2;
    int64 previous_block_height = 3;
}

message SideChainBlockData {
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_status_merkle_tree_root = 3;
    int32 chain_id = 4;
}

message ParentChainBlockData {
    int64 height = 1;
    CrossChainExtraData cross_chain_extra_data = 2;
    int32 chain_id = 3;
    aelf.Hash transaction_status_merkle_tree_root = 4;
```

(continues on next page)

(continued from previous page)

```
    map<int64, aelf.MerklePath> indexed_merkle_path = 5;
    map<string, bytes> extra_data = 6;
}

message CrossChainExtraData {
    aelf.Hash transaction_status_merkle_tree_root = 1;
}
```

Index block data of parent chain and side chain. Only **CrossChainIndexingController** is permitted to invoke this method.

Parameters:

RecordCrossChainDataInput

- **CrossChainBlockData**
  - **SideChainBlockData**
    - height : height of side chain block
    - block_header_hash : hash of side chain block
    - transaction_merkle_tree_root : merkle tree root computing from transactions status in side chain block
    - chain_id : id of side chain
  - **ParentChainBlockData**
    - height : height of parent chain
    - **cross_chain_extra_data**
      - transaction_status_merkle_tree_root : the merkle tree root computing from side chain roots.
    - chain_id : parent chain id
    - transaction_status_merkle_root : merkle tree root computing from transactions status in parent chain block
    - indexed_merkle_path : <block height, merkle path> key-value map
    - extra_data : extra data map

function AdjustIndexingFeePrice

```
rpc AdjustIndexingFeePrice(AdjustIndexingFeeInput)returns(google.protobuf.Empty){}

message AdjustIndexingFeeInput{
    int32 side_chain_id = 1;
    int64 indexing_fee = 2;
}
```

Adjust side chain indexing fee. Only **IndexingFeeController** is permitted to invoke this method.

Parameters:

AdjustIndexingFeeInput

- **side_chain_id** : side chain id
- **indexing_fee** : indexing fee to be set

function DisposeSideChain

```
rpc DisposeSideChain (google.protobuf.Int32Value) returns (google.protobuf.Int32Value) {}
```

Dispose the specified side chain. Only **SideChainLifetimeController** is permitted to invoke this method.

Parameters:

- **Int32Value** - the id of side chain to be disposed

Returns:

the id of disposed chain

function VerifyTransaction

```
rpc VerifyTransaction (VerifyTransactionInput) returns (google.protobuf.BoolValue)
{
    option (aelf.is_view) = true;
}
message VerifyTransactionInput {
    aelf.Hash transaction_id = 1;
    aelf.MerklePath path = 2;
    int64 parent_chain_height = 3;
    int32 verified_chain_id = 4;
}
```

Transaction cross chain verification.

Parameters:

VerifyTransactionInput

- **transaction_id** - transaction id
- **path** - merkle path for the transaction
- **parent_chain_height** - height of parent chain indexing this transaction
- **verified_chain_id** - id of the chain to be verified

Returns:

True if verification succeeded, otherwise false.

function LockedAddress

```
rpc GetSideChainCreator (google.protobuf.Int32Value) returns (aelf.Address) {
        option (aelf.is_view) = true;
}
```

Get side chain creator address.

Parameters:

- **Int32Value** - id of side chain

Returns:

Address of side chain creator.

function GetChainStatus

```
rpc GetChainStatus (google.protobuf.Int32Value) returns (GetChainStatusOutput) {
        option (aelf.is_view) = true;
}

message GetChainStatusOutput{
    SideChainStatus status = 1;
}

enum SideChainStatus
{
    FATAL = 0;
    ACTIVE = 1;
    INDEXING_FEE_DEBT = 2;
    TERMINATED = 3;
}
```

Gets the current status of the specified side chain.

Parameters:

- **Int32Value** - id of side chain.

Returns:

Current status of side chain.

- fatal: currently no meaning.
- active: the side-chain is being indexed.
- insufficient fee debt: debt for indexing fee to be payed off.
- terminated: the side chain cannot be indexed anymore.

function GetSideChainHeight

```
rpc GetSideChainHeight (google.protobuf.Int32Value) returns (google.protobuf.Int64Value)
↪{
        option (aelf.is_view) = true;
}
```

Get current height of the specified side chain.

Parameters:

- **Int32Value** - id of side chain

Returns:

Current height of the side chain.

function GetParentChainHeight

```
rpc GetParentChainHeight (google.protobuf.Empty) returns (google.protobuf.Int64Value) {
        option (aelf.is_view) = true;
}
```

Get recorded height of parent chain

Parameters:

- **google.protobuf.Empty**

Returns:

Height of parent chain.

function GetParentChainId

```
rpc GetParentChainId (google.protobuf.Empty) returns (google.protobuf.Int32Value) {
        option (aelf.is_view) = true;
}
```

Get id of the parent chain. This interface is only for side chain.

Parameters:

- **google.protobuf.Empty**

Returns:

Parent chain id.

function GetSideChainBalance

```
rpc GetSideChainBalance (google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value) {
    option (aelf.is_view) = true;
}
```

Get the balance for side chain indexing.

Parameters:

- **Int32Value** - id of side chain

Returns:

Balance for side chain indexing.

function GetSideChainIndexingFeeDebt

```
rpc GetSideChainIndexingFeeDebt (google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value) {
    option (aelf.is_view) = true;
}
```

Get indexing debt for side chain.

Parameters:

- **Int32Value** - id of side chain

Returns:

Side chain indexing debt. Returns zero if no debt.

function GetSideChainIdAndHeight

```
rpc GetSideChainIdAndHeight (google.protobuf.Empty) returns (SideChainIdAndHeightDict)
{
    option (aelf.is_view) = true;
}
message SideChainIdAndHeightDict
{
    map<int32, int64> id_height_dict = 1;
}
```

Get id and recorded height of side chains.

Parameters:

- **google.protobuf.Empty**

Returns:

SideChainIdAndHeightDict : A map contains id and height of side chains

function GetSideChainIndexingInformationList

```
rpc GetSideChainIndexingInformationList (google.protobuf.Empty) returns␣
↪(SideChainIndexingInformationList)
{
    option (aelf.is_view) = true;
}
message SideChainIndexingInformationList
{
    repeated SideChainIndexingInformation indexing_information_list = 1;
}
message SideChainIndexingInformation
{
    int32 chain_id = 1;
    int64 indexed_height = 2;
}
```

Get indexing information of side chains.

Parameters:

- **google.protobuf.Empty**

Returns:

SideChainIndexingInformationList : A list contains indexing information of side chains

function GetAllChainsIdAndHeight

```
rpc GetAllChainsIdAndHeight (google.protobuf.Empty) returns (SideChainIdAndHeightDict)
{
    option (aelf.is_view) = true;
}
message SideChainIdAndHeightDict
{
    map<int32, int64> id_height_dict = 1;
}
```

Get id and recorded height of all chains.

Parameters:

- **google.protobuf.Empty**

Returns:

SideChainIdAndHeightDict : A map contains id and height of all chains

function GetIndexedCrossChainBlockDataByHeight

```
rpc GetIndexedCrossChainBlockDataByHeight (google.protobuf.Int64Value) returns␣
↪(CrossChainBlockData) {
        option (aelf.is_view) = true;
}

message CrossChainBlockData {
    repeated SideChainBlockData side_chain_block_data_list = 1;
    repeated ParentChainBlockData parent_chain_block_data_list = 2;
    int64 previous_block_height = 3;
}

message SideChainBlockData {
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_status_merkle_tree_root = 3;
    int32 chain_id = 4;
}

message ParentChainBlockData {
    int64 height = 1;
    CrossChainExtraData cross_chain_extra_data = 2;
    int32 chain_id = 3;
    aelf.Hash transaction_status_merkle_tree_root = 4;
    map<int64, aelf.MerklePath> indexed_merkle_path = 5;
    map<string, bytes> extra_data = 6;
}

message CrossChainExtraData {
    aelf.Hash transaction_status_merkle_tree_root = 1;
}
```

Get indexed cross chain data by height.

Parameters:

- **Int64Value** - block height

Returns:

CrossChainBlockData

- **side_chain_block_data** - cross chain block data of side chain
- **parent_chain_block_data** - cross chain block data of parent chain

function GetIndexedSideChainBlockDataByHeight

```
rpc GetIndexedSideChainBlockDataByHeight (google.protobuf.Int64Value) returns␣
↪(IndexedSideChainBlockData) {
        option (aelf.is_view) = true;
}
message IndexedSideChainBlockData
{
    repeated acs7.SideChainBlockData side_chain_block_data = 1;
}
message SideChainBlockData
{
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_merkle_tree_root = 3;
    int32 chain_id = 4;
}
```

Get block data of indexed side chain by height

Parameters:

- **Int64Value** - height of side chain

Returns:

IndexedSideChainBlockData

- **side_chain_block_data** - block data of side chain

function GetBoundParentChainHeightAndMerklePathByHeight

```
rpc GetBoundParentChainHeightAndMerklePathByHeight (google.protobuf.Int64Value) returns␣
↪(CrossChainMerkleProofContext) {
        option (aelf.is_view) = true;
}
message CrossChainMerkleProofContext
{
    int64 bound_parent_chain_height = 1;
    aelf.MerklePath merkle_path_for_parent_chain_root = 2;
}
message MerklePath
{
    repeated MerklePathNode merklePathNodes = 1;
```

(continues on next page)

(continued from previous page)

```
}
message MerklePathNode
{
    Hash hash = 1;
    bool isLeftChildNode = 2;
}
```

Get merkle path bound up with side chain

Parameters:

- **Int64Value** - height of side chain

Returns:

CrossChainMerkleProofContext

- **bound_parent_chain_height** - height of parent chain bound up with side chain
- **merkle_path_from_parent_chain** - merkle path generated from parent chain

function GetChainInitializationData

```
rpc GetChainInitializationData (google.protobuf.Int32Value) returns␣
↪(ChainInitializationData) {
        option (aelf.is_view) = true;
}
message ChainInitializationData
{
    int32 chain_id = 1;
    aelf.Address creator = 2;
    google.protobuf.Timestamp creation_timestamp = 3;
    int64 creation_height_on_parent_chain = 4;
    bool chain_creator_privilege_preserved = 5;
    aelf.Address parent_chain_token_contract_address = 6;
    ChainInitializationConsensusInfo chain_initialization_consensus_info = 7;
    bytes native_token_info_data = 8;
    ResourceTokenInfo resource_token_info = 9;
    ChainPrimaryTokenInfo chain_primary_token_info = 10;
}

message ChainInitializationConsensusInfo{
    bytes initial_miner_list_data = 1;
}

message ResourceTokenInfo{
    bytes resource_token_list_data = 1;
    map<string, int32> initial_resource_amount = 2;
}

message ChainPrimaryTokenInfo{
    bytes chain_primary_token_data = 1;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 2;
}
```

(continues on next page)

(continued from previous page)

```
message SideChainTokenInitialIssue{
    aelf.Address address = 1;
    int64 amount = 2;
}
```

Get initialization data for specified side chain.

Parameters:

- **Int32Value** - id of side chain

Returns:

ChainInitializationData

- **chain_id** - id of side chain
- **creator** - side chain creator
- **creation_timestamp** - timestamp for side chain creation
- **creation_height_on_parent_chain** - height of side chain creation on parent chain
- **chain_creator_privilege_preserved** - creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.
- **parent_chain_token_contract_address** - parent chain token contract address
- **chain_initialization_consensus_info**
  - **initial_miner_list_data** consensus miner list data
- **native_token_info_data** - native token info
- **resource_token_info**
  - **resource_token_list_data** resource token list data
  - **initial_resource_amount** initial resource token amount
- **chain_primary_token_info**
  - **chain_primary_token_data** side chain primary token data
  - **side_chain_token_initial_issue_list** side chain primary token initial issue list

2.11 Treasury Contract

The Treasury contract is essentially used for distributing bonus' to voters and candidates during the election process.

2.11.1 Donate

Donates tokens from the caller to the treasury. If the tokens are not native tokens in the current chain, they will be first converted to the native token.

```
rpc Donate (DonateInput) returns (google.protobuf.Empty) {}

message DonateInput {
    string symbol = 1;
    sint64 amount = 2;
}

message DonationReceived {
    aelf.Address from = 1 [(aelf.is_indexed) = true];
    aelf.Address to = 2 [(aelf.is_indexed) = true];
    string symbol = 3 [(aelf.is_indexed) = true];
    sint64 amount = 4 [(aelf.is_indexed) = true];
    string memo = 5;
}
```

DonateInput:

- **symbol**: token symbol.
- **amount**: token amount.

After a successful donation a **DonationReceived** event log can be found in the transaction result.

DonationReceived:

- **from**: from address.
- **to**: to address.
- **symbol**: token symbol.
- **amount**: amount of token.
- **memo**: memo.

2.11.2 Donate all tokens

Donate all token (transfer to native token) from caller to the treasury (by calling **Donate** described above).

```
rpc DonateAll (DonateAllInput) returns (google.protobuf.Empty) {}

message DonateAllInput {
    string symbol = 1;
}
```

DonateAllInput:

- **symbol**: token symbol.

2.11.3 SetDistributingSymbolList

Set a token list that can be used to distribute.

```
rpc SetDistributingSymbolList (SymbolList) returns (google.protobuf.Empty) {}

message SymbolList {
```

(continues on next page)

(continued from previous page)

```
    repeated string value = 1;
}
```

SymbolList:

- **value**: token symbol list.

2.11.4 SetDividendPoolWeightSetting

Set weight for the three activities.

```
rpc SetDividendPoolWeightSetting (DividendPoolWeightSetting) returns (google.protobuf.
↪Empty){}

message DividendPoolWeightSetting {
    int32 citizen_welfare_weight = 1;
    int32 backup_subsidy_weight = 2;
    int32 miner_reward_weight = 3;
}
```

DividendPoolWeightSetting:

- **citizen welfare weight**: citizen welfare weight.
- **backup subsidy weight**: backup subsidy weight.
- **miner reward weight**: miner reward weight.

2.11.5 SetMinerRewardWeightSetting

Set weight for the three activities composing of miner reward activity.

```
rpc SetMinerRewardWeightSetting (MinerRewardWeightSetting) returns (google.protobuf.
↪Empty){}

message MinerRewardWeightSetting {
    int32 basic_miner_reward_weight = 1;
    int32 votes_weight_reward_weight = 2;
    int32 re_election_reward_weight = 3;
}
```

MinerRewardWeightSetting:

- **basic miner reward weight**: basic miner reward weight.
- **votes weight reward weight**: votes weight reward weight.
- **re-election reward weight**: re-election reward weight.

2.11.6 ChangeTreasuryController

Change the controller who is able to update symbol list and activities' weight above.

```
rpc ChangeTreasuryController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: controller type.
- **owner address**: controller's address.

2.11.7 view methods

For reference, you can find here the available view methods.

GetCurrentTreasuryBalance

Get the Treasury's total balance of the native token from the Treasury.

```
rpc GetCurrentTreasuryBalance (google.protobuf.Empty) returns (aelf.SInt64Value){}

message SInt64Value
{
    sint64 value = 1;
}
```

returns:

- **value**: amount of native token.

GetWelfareRewardAmountSample

Test the welfare bonus gotten base on 10000 Vote Token. The input is a array of locking time, and the output is the corresponding welfare.

```
rpc GetWelfareRewardAmountSample (GetWelfareRewardAmountSampleInput) returns␣
↪(GetWelfareRewardAmountSampleOutput) {}

message GetWelfareRewardAmountSampleInput {
    repeated sint64 value = 1;
}

message GetWelfareRewardAmountSampleOutput {
    repeated sint64 value = 1;
}
```

GetWelfareRewardAmountSampleInput:

- **value**: a array of locking time.

returns:

- **value**: a array of welfare.

GetTreasurySchemeId

Get treasury scheme id. If it does not exist, it will return hash.empty.

```
rpc GetTreasurySchemeId (google.protobuf.Empty) returns (aelf.Hash) {}

message Hash
{
    bytes value = 1;
}
```

returns:

- **value**: scheme id.

GetDistributingSymbolList

Get the symbol list that can be used to distribute.

```
rpc GetDistributingSymbolList (google.protobuf.Empty) returns (SymbolList) {}

message SymbolList {
    repeated string value = 1;
}
```

note: *for SymbolList see SetDistributingSymbolList*

GetDividendPoolWeightProportion

Get activities's weight expressed as a percentage

```
rpc GetDividendPoolWeightProportion (google.protobuf.Empty) returns␣
↪(DividendPoolWeightProportion){}

message DividendPoolWeightProportion {
    SchemeProportionInfo citizen_welfare_proportion_info = 1;
    SchemeProportionInfo backup_subsidy_proportion_info = 2;
    SchemeProportionInfo miner_reward_proportion_info = 3;
}

message SchemeProportionInfo{
    aelf.Hash scheme_id = 1;
    int32 proportion = 2;
}
```

returns:

- **citizen welfare proportion info**: citizen welfare proportion info.
- **backup subsidy proportion info**: backup subsidy proportion info.
- **miner reward proportion info**: miner reward proportion info.

SchemeProportionInfo:

- **scheme id**: scheme id

- **proportion**: the weight expressed as a percentage.

GetMinerRewardWeightProportion

Get the weight expressed as a percentage of the activities composing of miner reward.

```
rpc GetMinerRewardWeightProportion (google.protobuf.Empty) returns␣
↪(MinerRewardWeightProportion){}

message MinerRewardWeightProportion {
    SchemeProportionInfo basic_miner_reward_proportion_info = 1;
    SchemeProportionInfo votes_weight_reward_proportion_info = 2;
    SchemeProportionInfo re_election_reward_proportion_info = 3;
}
```

note: *for MinerRewardWeightProportion see GetDividendPoolWeightProportion*

GetTreasuryController

Get this contract's controller.

```
rpc GetTreasuryController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

note: *for AuthorityInfo see ChangeTreasuryController*

2.12 Vote Contract

The Vote contract is an abstract layer for voting. Developers implement concrete voting activities by calling this contract.

2.12.1 Voting for Block Producers

To build a voting activity, the developer should register first.

```
rpc Register (VotingRegisterInput) returns (google.protobuf.Empty){
}

message VotingRegisterInput {
    google.protobuf.Timestamp start_timestamp = 1;
    google.protobuf.Timestamp end_timestamp = 2;
    string accepted_currency = 3;
    bool is_lock_token = 4;
    sint64 total_snapshot_number = 5;
    repeated string options = 6;
}
```

VotingRegisterInput:

- **start timestamp**: activity start time.
- **end timestamp**: activity end time.
- **accepted currency**: the token symbol which will be accepted.
- **is lock token**: indicates whether the token will be locked after voting.
- **total snapshot number**: number of terms.
- **options**: default candidate.

2.12.2 Vote

After building successfully a voting activity, others are able to vote.

```
rpc Vote (VoteInput) returns (google.protobuf.Empty){
}

message VoteInput {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 amount = 3;
    string option = 4;
    aelf.Hash vote_id = 5;
    bool is_change_target = 6;
}

message Voted {
    option (aelf.is_event) = true;
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp vote_timestamp = 5;
    string option = 6;
    aelf.Hash vote_id = 7;
}
```

VoteInput:

- **voting item id**: indicates which voting activity the user participate in.
- **voter**: voter's address.
- **amount**: vote amount.
- **option**: candidate's public key.
- **vote id**: transaction id.
- **is change target**: indicates whether the option is changed.

After a successfully vote, a **Voted** event log can be found in the transaction result.

Voted:

- **voting item id**: voting activity id.

- **voter**: voter's address.
- **snapshot number**: the current round.
- **amount**: vote amount.
- **vote timestamp**: vote time.
- **option**: the candidate's public key.
- **vote id**: transaction id.

2.12.3 Withdraw

A voter can withdraw the token after the lock time.

```
rpc Withdraw (WithdrawInput) returns (google.protobuf.Empty){
}

message WithdrawInput {
    aelf.Hash vote_id = 1;
}

message Withdrawn {
    aelf.Hash vote_id = 1;
}
```

WithdrawInput:

- **vote id**: transaction id.

After a successful vote, a **Withdrawn** event log can be found in the transaction result.

Withdrawn:

- **vote id**: transaction id.

2.12.4 TakeSnapshot

Distributes profits and saves the state every round.

```
rpc TakeSnapshot (TakeSnapshotInput) returns (google.protobuf.Empty) {}

message TakeSnapshotInput {
    aelf.Hash voting_item_id = 1;
    sint64 snapshot_number = 2;
}
```

TakeSnapshotInput:

- **voting item id**: voting activity id.
- **snapshot number**: the round number.

2.12.5 AddOption

Adds an option (a choice) to a voting activity.

```
rpc AddOption (AddOptionInput) returns (google.protobuf.Empty){
}

message AddOptionInput {
    aelf.Hash voting_item_id = 1;
    string option = 2;
}
```

AddOptionInput:

- **voting item id**: vote activity id.
- **option**: the new option.

2.12.6 AddOptions

Adds multiple options (choices) to a voting activity.

```
rpc AddOptions (AddOptionsInput) returns (google.protobuf.Empty){
}

message AddOptionsInput {
    aelf.Hash voting_item_id = 1;
    repeated string options = 2;
}
```

AddOptionsInput:

- **voting item id**: voting activity id.
- **option**: the list of new options.

2.12.7 RemoveOption

Removes an option from a voting activity.

```
rpc RemoveOption (RemoveOptionInput) returns (google.protobuf.Empty){
}

message RemoveOptionInput {
    aelf.Hash voting_item_id = 1;
    string option = 2;
}
```

RemoveOptionInput:

- **voting item id**: voting activity id.
- **option**: the option to remove.

2.12.8 RemoveOptions

Removes multiple options from a voting activity.

```
rpc RemoveOptions (RemoveOptionsInput) returns (google.protobuf.Empty){
}

message RemoveOptionsInput {
    aelf.Hash voting_item_id = 1;
    repeated string options = 2;
}
```

RemoveOptionsInput:

- **voting item id**: voting activity id.
- **option**: the options to remove.

2.12.9 view methods

For reference, you can find here the available view methods.

GetVotingItem

Gets the information related to a voting activity.

```
rpc GetVotingItem (GetVotingItemInput) returns (VotingItem){
}

message GetVotingItemInput {
    aelf.Hash voting_item_id = 1;
}

message VotingItem {
    aelf.Hash voting_item_id = 1;
    string accepted_currency = 2;
    bool is_lock_token = 3;
    sint64 current_snapshot_number = 4;
    sint64 total_snapshot_number = 5;
    repeated string options = 6;
    google.protobuf.Timestamp register_timestamp = 7;
    google.protobuf.Timestamp start_timestamp = 8;
    google.protobuf.Timestamp end_timestamp = 9;
    google.protobuf.Timestamp current_snapshot_start_timestamp = 10;
    aelf.Address sponsor = 11;
}
```

GetVotingItemInput:

- **voting item id**: voting activity id.

returns:

- **voting item id**: voting activity id.
- **accepted currency**: vote token.
- **is lock token**: indicates if the token will be locked after voting.
- **current snapshot number**: current round.

- **total snapshot number**: total number of round.
- **register timestamp**: register time.
- **start timestamp**: start time.
- **end timestamp**: end time.
- **current snapshot start timestamp**: current round start time.
- **sponsor**: activity creator.

GetVotingResult

Gets a voting result according to the provided voting activity id and round number.

```
rpc GetVotingResult (GetVotingResultInput) returns (VotingResult) {}

message GetVotingResultInput {
    aelf.Hash voting_item_id = 1;
    sint64 snapshot_number = 2;
}

message VotingResult {
    aelf.Hash voting_item_id = 1;
    map<string, sint64> results = 2; // option -> amount
    sint64 snapshot_number = 3;
    sint64 voters_count = 4;
    google.protobuf.Timestamp snapshot_start_timestamp = 5;
    google.protobuf.Timestamp snapshot_end_timestamp = 6;
    sint64 votes_amount = 7;
}
```

GetVotingResultInput:

- **voting item id**: voting activity id.
- **snapshot number**: round number.

returns:

- **voting item id**: voting activity id.
- **results**: candidate => vote amount.
- **snapshot number**: round number.
- **voters count**: how many voters.
- **snapshot start timestamp**: start time.
- **snapshot end timestamp**: end time.
- **votes amount** total votes(excluding withdraws).

GetLatestVotingResult

Gets the latest result of the provided voting activity.

```
rpc GetLatestVotingResult (aelf.Hash) returns (VotingResult) {}

message Hash
{
    bytes value = 1;
}

message VotingResult {
    aelf.Hash voting_item_id = 1;
    map<string, sint64> results = 2; // option -> amount
    sint64 snapshot_number = 3;
    sint64 voters_count = 4;
    google.protobuf.Timestamp snapshot_start_timestamp = 5;
    google.protobuf.Timestamp snapshot_end_timestamp = 6;
    sint64 votes_amount = 7;
}
```

Hash:

- **value**: voting activity id.

returns:

- **voting item id**: voting activity id.
- **results**: candidate => vote amount.
- **snapshot number**: round number.
- **voters count**: how many voters.
- **snapshot start timestamp**: start time.
- **snapshot end timestamp**: end time.
- **votes amount**: total votes(excluding withdraws).

GetVotingRecord

Get the voting record for the given record ID.

```
rpc GetVotingRecord (aelf.Hash) returns (VotingRecord){
}

message Hash{
    bytes value = 1;
}

message VotingRecord {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp withdraw_timestamp = 5;
    google.protobuf.Timestamp vote_timestamp = 6;
    bool is_withdrawn = 7;
```

(continues on next page)

(continued from previous page)

```
    string option = 8;
    bool is_change_target = 9;
}
```

Hash:

- **value**: transaction id.

returns:

- **voting item id**: voting activity id.
- **voter**: voter's address.
- **snapshot number**: round number.
- **withdraw timestamp**: withdraw time.
- **vote timestamp**: vote time.
- **is withdrawn**: indicate whether the vote has been withdrawn.
- **option**: candidate id.
- **is change target**: has withdrawn and vote to others.

GetVotingRecords

Get the voting records for the given record IDs.

```
rpc GetVotingRecords (GetVotingRecordsInput) returns (VotingRecords){
}

message GetVotingRecordsInput {
    repeated aelf.Hash ids = 1;
}

message Hash
{
    bytes value = 1;
}

message VotingRecords {
    repeated VotingRecord records = 1;
}

message VotingRecord {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp withdraw_timestamp = 5;
    google.protobuf.Timestamp vote_timestamp = 6;
    bool is_withdrawn = 7;
    string option = 8;
    bool is_change_target = 9;
}
```

GetVotingRecordsInput:

- **ids**: transaction ids.

Hash:

- **value**: transaction id.

returns:

- **records**: records.

VotingRecord:

- **voting item id**: voting activity id.
- **voter**: voter's address.
- **snapshot number**: round number.
- **withdraw timestamp**: withdraw time.
- **vote timestamp**: vote time.
- **is withdrawn**: indicates whether the vote has been withdrawn.
- **option**: candidate id.
- **is change target**: has withdrawn and vote to others.

GetVotedItems

Get the voter's withdrawn and valid transaction ids respectively.

```
rpc GetVotedItems (aelf.Address) returns (VotedItems){
}

message Address{
    bytes value = 1;
}

message VotedItems {
    map<string, VotedIds> voted_item_vote_ids = 1;
}

message VotedIds {
    repeated aelf.Hash active_votes = 1;
    repeated aelf.Hash withdrawn_votes = 2;
}
```

Address:

- **value**: voter's address.

returns:

- **voted item vote ids**: voting activity id => vote information.

VotedIds:

- **active votes**: valid transaction id.
- **withdrawn votes**: withdrawn transaction id.

GetVotingIds

Get the voter's withdrawn and valid transaction ids respectively according to voting activity id.

```
rpc GetVotingIds (GetVotingIdsInput) returns (VotedIds){
}

message GetVotingIdsInput {
    aelf.Address voter = 1;
    aelf.Hash voting_item_id = 2;
}

message VotedIds {
    repeated aelf.Hash active_votes = 1;
    repeated aelf.Hash withdrawn_votes = 2;
}
```

GetVotingIdsInput:

- **voter**: voter's address.
- **voting item id**: voting activity id.

returns:

- **active votes**: valid transaction id.
- **withdrawn votes**: withdrawn transaction id.

2.13 Token Holder Contract

The TokenHolder contract is essentially used for building a bouns model for distributing bonus' to whom hold the token.

2.13.1 CreateScheme

```
rpc CreateScheme (CreateTokenHolderProfitSchemeInput) returns (google.protobuf.Empty) { }

message CreateTokenHolderProfitSchemeInput {
    string symbol = 1;
    sint64 minimum_lock_minutes = 2;
    map<string, sint64> auto_distribute_threshold = 3;
}
```

CreateTokenHolderProfitSchemeInput:

- **symbol**: the token that will be used for locking and distributing profits.
- **minimum** lock time: minimum lock time before withdrawing.
- **automatic distribution threshold**: used when registering for profits (RegisterForProfits).

2.13.2 AddBeneficiary

```
rpc AddBeneficiary (AddTokenHolderBeneficiaryInput) returns (google.protobuf.Empty) { }

message AddTokenHolderBeneficiaryInput {
    aelf.Address beneficiary = 1;
    sint64 shares = 2;
}
```

AddTokenHolderBeneficiaryInput:

- **beneficiary**: the new beneficiary.
- **shares**: the shares to attribute to this beneficiary.

2.13.3 RemoveBeneficiary

```
rpc RemoveBeneficiary (RemoveTokenHolderBeneficiaryInput) returns (google.protobuf.
↪Empty) { }

message RemoveTokenHolderBeneficiaryInput {
    aelf.Address beneficiary = 1;
    sint64 amount = 2;
}
```

Note: this method can be used to remove a beneficiary or update its shares.

RemoveTokenHolderBeneficiaryInput:

- **beneficiary**: the beneficiary to remove or update.
- **amount**: 0 to remove the beneficiary. A positive integer, smaller than the current shares.

2.13.4 ContributeProfits

```
rpc ContributeProfits (ContributeProfitsInput) returns (google.protobuf.Empty) { }

message ContributeProfitsInput {
    aelf.Address scheme_manager = 1;
    sint64 amount = 2;
    string symbol = 3;
}
```

ContributeProfitsInput:

- **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
- **amount**: the amount of tokens to contribute.
- **symbol**: the token to contribute.

2.13.5 DistributeProfits

```
rpc DistributeProfits (DistributeProfitsInput) returns (google.protobuf.Empty) { }

message DistributeProfitsInput {
    aelf.Address scheme_manager = 1;
    string symbol = 2;
}
```

DistributeProfitsInput:

- **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
- **symbol**: the token to contribute.

2.13.6 RegisterForProfits

```
rpc RegisterForProfits (RegisterForProfitsInput) returns (google.protobuf.Empty) { }

message RegisterForProfitsInput {
    aelf.Address scheme_manager = 1;
    sint64 amount = 2;
}
```

RegisterForProfitsInput:

- **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
- **amount**: the amount of tokens to lock (and will correspond to the amount of shares).

2.13.7 Withdraw

```
rpc Withdraw (aelf.Address) returns (google.protobuf.Empty) { }
```

This method will withdraw the given address for the Token Holder contract, this will also unlock the previously locked tokens.

2.13.8 ClaimProfits

```
rpc ClaimProfits (ClaimProfitsInput) returns (google.protobuf.Empty) { }

message ClaimProfitsInput {
    aelf.Address scheme_manager = 1;
    aelf.Address beneficiary = 2;
    string symbol = 3;
}
```

ClaimProfitsInput:

- **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
- **beneficiary**: the beneficiary, defaults to the Sender.
- **symbol**: the symbol to claim.

2.13.9 View methods

2.13.10 GetScheme

```
rpc GetScheme (aelf.Address) returns (TokenHolderProfitScheme) { }

message TokenHolderProfitScheme {
    string symbol = 1;
    aelf.Hash scheme_id = 2;
    sint64 period = 3;
    sint64 minimum_lock_minutes = 4;
    map<string, sint64> auto_distribute_threshold = 5;
}
```

Returns a description of the scheme, wrapped in a **TokenHolderProfitScheme** object:

- **symbol**: the scheme's token.
- **scheme id**: the id of the scheme.
- **period**: the current period of the scheme.
- **minimum lock minutes**: minimum lock time.
- **automatic distribution threshold**: distribution info.

2.14 Economic Contract

The Economic contract establishes the economic system of the AElf. When the block chain starts to work, this contract will initialize other contracts related to economic activities.

2.14.1 InitialEconomicSystem

It will initialize other contracts related to economic activities (For instance, create the native token). This transaction only can be send once because after the first sending, its state will be set to initialized.

```
rpc InitialEconomicSystem (InitialEconomicSystemInput) returns (google.protobuf.Empty) {
}

message InitialEconomicSystemInput {
    string native_token_symbol = 1;
    string native_token_name = 2;
    int64 native_token_total_supply = 3;
    int32 native_token_decimals = 4;
    bool is_native_token_burnable = 5;
    int64 mining_reward_total_amount = 6;
    int64 transaction_size_fee_unit_price = 7;
}
```

IssueNativeTokenInput:

- **native token symbol**: the native token symbol.
- **native token name**: the native token name.

- **native token total supply**: the native token total supply.
- **native token decimals**: the token calculation is accurated to which decimal.
- **is native token burnable**: it indicaites if the token is burnable.
- **mining reward total amount**: It determines how much native token is used to reward the miners.
- **transaction size fee unit price**: the transaction fee = transaction size * unit fee.

2.14.2 IssueNativeToken

Only ZeroContract is able to issue the native token.

```
rpc IssueNativeToken (IssueNativeTokenInput) returns (google.protobuf.Empty) {
}

message IssueNativeTokenInput {
    int64 amount = 1;
    string memo = 2;
    aelf.Address to = 3;
}
```

IssueNativeTokenInput:

- **amount**: amount of token.
- **memo**: memo.
- **to**: the recipient of the token.

2.14.3 ACS1 Implementation

For reference, you can find here the methods implementing acs1.

SetMethodFee

It sets method fee.

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

MethodFees:

- **method name** the method name in this contract.

- **fees** fee list.

MethodFee:

- **symbol** token symbol.
- **basic fee** fee.

ChangeMethodFeeController

Change the method fee controller, the default is Parliament.

```
rpc ChangeMethodFeeController (AuthorityInfo) returns (google.protobuf.Empty) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

GetMethodFee

This mehtod is used to query the method fee information.

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees) {
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

This mehtod is used to query the method fee information.

```
rpc GetMethodFeeController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

note: *for AuthorityInfo see ChangeMethodFeeController*

2.15 TokenConvert Contract

Using this contract can build a connection between the base token(the default is native token) and other tokens created on the chain. After building the connection, users can trade tokens with the Bancor model. You can find the detail information about Bancor in AElf Economic System White Paper.

2.15.1 InitializeInput

This method is used to initialize this contract (add base token, connections, etc.).

```
rpc Initialize (InitializeInput) returns (google.protobuf.Empty) {
}

message InitializeInput {
    string base_token_symbol = 1;
    string fee_rate = 2;
    repeated Connector connectors = 3;
}

message Connector {
    string symbol = 1;
    int64 virtual_balance = 2;
    string weight = 3;
    bool is_virtual_balance_enabled = 4;
    bool is_purchase_enabled = 5;
    string related_symbol = 6;
    bool is_deposit_account = 7;
}
```

InitializeInput:

- **base token symbol**: base token, default is the native token.
- **fee rate**: buy/sell token need pay the fee(= cost * feeRate).
- **connectors**: the default added connectors.

Connector:

We use Bancor model to build the connection between base token and other tokens. Each pair connectors include the coefficients used by calculating the token price based on the base token, and it consists of the base token connector and the new token connector.

- **symbol**: the connector symbol.
- **related symbol**: indicates its related connector, the pair connector includes a new created token connector and the base token connector.
- **virtual balance**: used in bancor model.
- **weight**: the weight is linked to the related connector's weight.
- **is virtual balance enabled**: true indicates that the virtual balance is used in price calculation.
- **is purchase enabled**: after build a pair connector, you can not buy/sell the token immediately, the default is false.
- **is deposit account**: indicates if the connector is base token connector.

2.15.2 AddPairConnector

With Bancor model, each new token need a pair connectors to calculate its price. Only the connector contoller(the default is parliament) is allowed to call this API.

```
rpc AddPairConnector(PairConnectorParam) returns (google.protobuf.Empty){
}

message PairConnectorParam {
    string resource_connector_symbol = 1;
    string resource_weight = 2;
    int64 native_virtual_balance = 3;
    string native_weight = 4;
}
```

PairConnectorParam:

- **resource connector symbol**: the new token connector's symbol.
- **resource weight**: the new token connector's weight.
- **native virtual balance**: the related base token connector's virtual balance.
- **native weight**: base token's weight.

2.15.3 EnableConnector

To make the connection work, the connector contoller need send this transaction.

```
rpc EnableConnector (ToBeConnectedTokenInfo) returns (google.protobuf.Empty) {
}

message ToBeConnectedTokenInfo{
    string token_symbol = 1;
    int64 amount_to_token_convert = 2;
}
```

ToBeConnectedTokenInfo:

- **token symbol**: the token symbol.
- **amount to token convert**: to make the token trade, maybe you need deposit some base token.

2.15.4 UpdateConnector

Before calling the EnableConnector, the connector contoller update the pair connctors' information.

```
rpc UpdateConnector(Connector) returns (google.protobuf.Empty){
}
```

note: *for Connector see Initialize*

2.15.5 Buy

After building the connection and enabling it, you can buy the new token with the base token.

```
rpc Buy (BuyInput) returns (google.protobuf.Empty) {
}

message BuyInput {
    string symbol = 1;
    int64 amount = 2;
    int64 pay_limit = 3;
}
```

BuyInput:

- **symbol**: the token symbol.
- **amount**: the amount you want to buy.
- **pay limit**: no buy if paying more than this, 0 if no limit.

2.15.6 Sell

After building the connection and enabling it, you can sell the new token.

```
rpc Sell (SellInput) returns (google.protobuf.Empty) {
}

message SellInput {
    string symbol = 1;
    int64 amount = 2;
    int64 receive_limit = 3;
}
```

SellInput:

- **symbol**: the token symbol.
- **amount**: the amount you want to sell.
- **receive limit**: no sell if receiving less than this, 0 if no limit.

2.15.7 ChangeConnectorController

The controller can be transfer to others.

```
rpc ChangeConnectorController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

2.15.8 ACS1 Implementation

For reference, you can find here the methods implementing acs1.

SetMethodFee

It sets method fee.

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

MethodFees:

- **method name** the method name in this contract.
- **fees** fee list.

MethodFee:

- **symbol** token symbol.
- **basic fee** fee.

ChangeMethodFeeController

Change the method fee controller, the default is Parliament.

```
rpc ChangeMethodFeeController (AuthorityInfo) returns (google.protobuf.Empty) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

GetMethodFee

This mehtod is used to query the method fee information.

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees) {
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

This mehtod is used to query the method fee information.

```
rpc GetMethodFeeController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

note: *for AuthorityInfo see ChangeMethodFeeController*

2.15.9 view methods

For reference, you can find here the available view methods.

GetFeeReceiverAddress

This method is used to query a the fee receiver.

```
rpc GetFeeReceiverAddress (google.protobuf.Empty) returns (aelf.Address) {
}

message Address{
    bytes value = 1;
}
```

returns:

- **value** the receiver's address, the half fee is burned and the other half is transferred to receiver.

GetControllerForManageConnector

This method is used to query the controller.

```
rpc GetControllerForManageConnector (google.protobuf.Empty) returns (acs1.AuthorityInfo)
↪{
}
```

(continues on next page)

(continued from previous page)

```
message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

note: *for AuthorityInfo see ChangeConnectorController*

GetPairConnector

This method is used to get the connection information between the base token and other token, which inlcudes a pair connctors.

```
rpc GetPairConnector (TokenSymbol) returns (PairConnector) {
}

message TokenSymbol {
    string symbol = 1;
}

message PairConnector{
    Connector resource_connector = 1;
    Connector deposit_connector = 2;
}
```

TokenSymbol:

- **symbol**: the token symbol.

returns:

- **resource connector**: the new add token connector.
- **deposit connector**: the corresponding base token connector.

note: *for Connector see Initialize*

GetFeeRate

This method is used to query the fee rate.

```
rpc GetFeeRate (google.protobuf.Empty) returns (google.protobuf.StringValue) {
}

message StringValue {
  string value = 1;
}
```

returns:

- **value**: the fee rate.

GetBaseTokenSymbol

This method is used to query the base token symbol.

```
rpc GetBaseTokenSymbol (google.protobuf.Empty) returns (TokenSymbol) {
}

message TokenSymbol {
  string symbol = 1;
}
```

returns:

- **symbol**: the token symbol.

GetBaseGetNeededDeposit

This method is used to query how much the base token need be deposited before enabling the connectors.

```
rpc GetNeededDeposit(ToBeConnectedTokenInfo) returns (DepositInfo) {
}

message ToBeConnectedTokenInfo{
    string token_symbol = 1;
    int64 amount_to_token_convert = 2;
}

message DepositInfo{
    int64 need_amount = 1;
    int64 amount_out_of_token_convert = 2;
}
```

ToBeConnectedTokenInfo:

- **token symbol**: the token symbol.
- **amount to token convert**: the added token amount you decide to transfer to TokenConvert.

returns:

- **need amount**: besides the amount you transfer to TokenConvert, how much base token you need deposit.
- **amount out of token convert**: how much the added token have not transferred to TokenConvert.

GetDepositConnectorBalance

This method is used to query how much the base token have been deposited.

```
rpc GetDepositConnectorBalance(google.protobuf.StringValue) returns (google.protobuf.
↪Int64Value){
}

message StringValue {
  string value = 1;
}

message Int64Value {
```

(continues on next page)

(continued from previous page)

```
  int64 value = 1;
}
```

StringValue:

- **value**: the token symbol.

returns:

- **value**: indicates for this token how much the base token have been deposited.

2.16 Configuration Contract

This contract's controller(the default is parliament) is able to save data(configuration) on the block chain.

2.16.1 SetConfiguration

This method is used to add or reset configurations.

```
rpc SetConfiguration (SetConfigurationInput) returns (google.protobuf.Empty) {
}

message SetConfigurationInput {
    string key = 1;
    bytes value = 2;
}
```

SetConfigurationInput:

- **key**: the configuration's key.
- **value**: the configuration's value(bianry data).

2.16.2 ChangeConfigurationController

The controller can be transfer to others.

```
rpc ChangeConfigurationController (acs1.AuthorityInfo) returns (google.protobuf.Empty) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

2.16.3 ACS1 Implementation

For reference, you can find here the methods implementing acs1.

SetMethodFee

It sets method fee.

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

MethodFees:

- **method name** the method name in this contract.
- **fees** fee list.

MethodFee:

- **symbol** token symbol.
- **basic fee** fee.

ChangeMethodFeeController

Change the method fee controller, the default is Parliament.

```
rpc ChangeMethodFeeController (AuthorityInfo) returns (google.protobuf.Empty) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

AuthorityInfo:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

GetMethodFee

This mehtod is used to query the method fee information.

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees) {
}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

This mehtod is used to query the method fee information.

```
rpc GetMethodFeeController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

note: *for AuthorityInfo see ChangeMethodFeeController*

2.16.4 view methods

For reference, you can find here the available view methods.

GetConfiguration

This method is used to query a configurations.

```
rpc GetConfiguration (google.protobuf.StringValue) returns (google.protobuf.BytesValue) {
}

message StringValue {
  string value = 1;
}

message BytesValue {
  bytes value = 1;
}
```

StringValue:

- **value**: the configuration's key.

returns:

- **value** the configuration's data(bianry).

GetConfigurationController

This method is used to query the controller information.

```
rpc GetConfigurationController (google.protobuf.Empty) returns (acs1.AuthorityInfo) {
}

message AuthorityInfo {
    aelf.Address contract_address = 1;
    aelf.Address owner_address = 2;
}
```

returns:

- **contract address**: new controller's contract address.
- **owner address**: new controller's address.

CHAPTER 3

Acs Introduction

3.1 ACS1 - Transaction Fee Standard

ACS1 is used to manage the transfer fee.

3.1.1 Interface

The contract inherited from ACS1 need implement the APIs below:

- SetMethodFee, the parameter type MethodFees defined in acs1.proto indicates the method name and its fee.
- GetMethodFee is used to get the method fee according to your input(method name).
- ChangeMethodFeeController is used to set who has the ablitity to call SetMethodFee, and its parameter's type AuthorityInfo is defined in authority_info.proto.
- GetMehodFeeController is used to get who has the ablitity to call SetMethodFee.

Attention: just the system contract on main chain is able to implement acs1.

3.1.2 Usage

On AElf main chain, before a transactoin start to execute, a pre-transaction is generated by pre-plugin FeeChargePreExecutionPlugin. It is used to charge the transaction fee.

The generated transaction's method is ChargeTransactionFees. The implementation is roughly like that (part of the code is omitted):

```
/// <summary>
/// Related transactions will be generated by acs1 pre-plugin service,
/// and will be executed before the origin transaction.
/// </summary>
```

(continues on next page)

(continued from previous page)

```
/// <param name="input"></param>
/// <returns></returns>
public override BoolValue ChargeTransactionFees(ChargeTransactionFeesInput input)
{
    // ...
    // Record tx fee bill during current charging process.
    var bill = new TransactionFeeBill();
    var fromAddress = Context.Sender;
    var methodFees = Context.Call<MethodFees>(input.ContractAddress,␣
↪nameof(GetMethodFee),
        new StringValue {Value = input.MethodName});
    var successToChargeBaseFee = true;
    if (methodFees != null && methodFees.Fees.Any())
    {
        successToChargeBaseFee = ChargeBaseFee(GetBaseFeeDictionary(methodFees), ref␣
↪bill);
    }
    var successToChargeSizeFee = true;
    if (!IsMethodFeeSetToZero(methodFees))
    {
        // Then also do not charge size fee.
        successToChargeSizeFee = ChargeSizeFee(input, ref bill);
    }
    // Update balances.
    foreach (var tokenToAmount in bill.FeesMap)
    {
        ModifyBalance(fromAddress, tokenToAmount.Key, -tokenToAmount.Value);
        Context.Fire(new TransactionFeeCharged
        {
            Symbol = tokenToAmount.Key,
            Amount = tokenToAmount.Value
        });
        if (tokenToAmount.Value == 0)
        {
            //Context.LogDebug(() => $"Maybe incorrect charged tx fee of {tokenToAmount.
↪Key}: it's 0.");
        }
    }
    return new BoolValue {Value = successToChargeBaseFee && successToChargeSizeFee};
}
```

In this method, the transaction fee consists of two parts:

1. The system calls GetMethodFee(line 15) to get the transacion fee you should pay. Then, it will check whether your balance is enough. If your balance is sufficient, the fee will be signed in the bill (variant bill). If not, your transaction will be rejected.
2. If the method fee is not set to 0 by the contract developer, the system will charge size fee. (the size if calculate by the paramter's size)

After charging successfully, an TransactionFeeCharged event is thrown, and the balance of the sender is modified.

The TransactionFeeCharged event will be captured and processed on the chain to calculate the total amount of transaction fees charged in the block. In the next block, the 10% of the transaction fee charged in this

block is destroyed, the remaining 90% flows to dividend pool on the main chain, and is transferred to the FeeReciever on the side chain. The code is:

```
/// <summary>
/// Burn 10% of tx fees.
/// If Side Chain didn't set FeeReceiver, burn all.
/// </summary>
/// <param name="symbol"></param>
/// <param name="totalAmount"></param>
private void TransferTransactionFeesToFeeReceiver(string symbol, long totalAmount)
{
    Context.LogDebug(() => "Transfer transaction fee to receiver.");
    if (totalAmount <= 0) return;
    var burnAmount = totalAmount.Div(10);
    if (burnAmount > 0)
        Context.SendInline(Context.Self, nameof(Burn), new BurnInput
        {
            Symbol = symbol,
            Amount = burnAmount
        });
    var transferAmount = totalAmount.Sub(burnAmount);
    if (transferAmount == 0)
        return;
    var treasuryContractAddress =
        Context.GetContractAddressByName(SmartContractConstants.
↪TreasuryContractSystemName);
    if ( treasuryContractAddress!= null)
    {
        // Main chain would donate tx fees to dividend pool.
        if (State.DividendPoolContract.Value == null)
            State.DividendPoolContract.Value = treasuryContractAddress;
        State.DividendPoolContract.Donate.Send(new DonateInput
        {
            Symbol = symbol,
            Amount = transferAmount
        });
    }
    else
    {
        if (State.FeeReceiver.Value != null)
        {
            Context.SendInline(Context.Self, nameof(Transfer), new TransferInput
            {
                To = State.FeeReceiver.Value,
                Symbol = symbol,
                Amount = transferAmount,
            });
        }
        else
        {
            // Burn all!
            Context.SendInline(Context.Self, nameof(Burn), new BurnInput
            {
```

(continues on next page)

(continued from previous page)

```
                Symbol = symbol,
                Amount = transferAmount
            });
        }
    }
}
```

In this way, AElf charges the transaction fee via the GetMethodFee provided by ACS1, and the other three methods are used to help with the implementations of GetMethodFee.

3.1.3 Implementation

The easiest way to do this is to just implement the method GetMethodFee.

If there are Foo1, Foo2, Bar1 and Bar2 methods related to business logic in a contract, they are priced as 1, 1, 2, 2 ELF respectively, and the transaction fees of these four methods will not be easily modified later, they can be implemented as follows:

```
public override MethodFees GetMethodFee(StringValue input)
{
    if (input.Value == nameof(Foo1) || input.Value == nameof(Foo2))
    {
        return new MethodFees
        {
            MethodName = input.Value,
            Fees =
            {
                new MethodFee
                {
                    BasicFee = 1_00000000,
                    Symbol = Context.Variables.NativeSymbol
                }
            }
        };
    }
    if (input.Value == nameof(Bar1) || input.Value == nameof(Bar2))
    {
        return new MethodFees
        {
            MethodName = input.Value,
            Fees =
            {
                new MethodFee
                {
                    BasicFee = 2_00000000,
                    Symbol = Context.Variables.NativeSymbol
                }
            }
        };
    }
    return new MethodFees();
}
```

This implementation can modify the transaction fee only by upgrading the contract, without implementing the other three interfaces.

A more recommended implementation needs to define an MappedState in the State file for the contract:

```
public MappedState<string, MethodFees> TransactionFees { get; set; }
```

Modify the TransactionFees data structure in the SetMethodFee method, and return the value in the GetMethodFee method.

In this solution, the implementation of GetMethodFee is very easy:

```
public override MethodFees GetMethodFee(StringValue input)
    return State.TransactionFees[input.Value];
}
```

The implementation of SetMethodFee requires the addition of permission management, since contract developers don't want the transaction fees of their contract methods to be arbitrarily modified by others.

Referring to the MultiToken contract, it can be implemented as follows:

Firstly, define a SingletonState AuthorityInf(the AuthorityInf is defined in authority_info.proto)

```
public SingletonState<AuthorityInfo> MethodFeeController { get; set; }
```

Then, check the sender's right by comparing its address with OwnerAdress.

```
public override Empty SetMethodFee(MethodFees input)
{
  foreach (var symbolToAmount in input.Fees)
  {
     AssertValidToken(symbolToAmount.Symbol, symbolToAmount.BasicFee);
  }
  RequiredMethodFeeControllerSet();
  Assert(Context.Sender ==             State.MethodFeeController.Value.OwnerAddress,
↪"Unauthorized to set method fee.");
    State.TransactionFees[input.MethodName] = input;
    return new Empty();
}
```

AssertValidToken checks if the token symbol exists, and the BasicFee is reasonable.

The permission check code is in the lines 8 and 9, and RequiredMethodFeeControllerSet prevents the permission is not set before.

If permissions are not set, the SetMethodFee method can only be called by the default address of the Parliamentary contract. If a method is sent through the default address of Parliament, it means that two-thirds of the block producers have agreed to the proposal.

```
private void RequiredMethodFeeControllerSet()
{
   if (State.MethodFeeController.Value != null) return;
   if (State.ParliamentContract.Value == null)
   {
     State.ParliamentContract.Value =         Context.
↪GetContractAddressByName(SmartContractConstants.ParliamentContractSystemName);
   }
```

(continues on next page)

(continued from previous page)

```
    var defaultAuthority = new AuthorityInfo();
    // Parliament Auth Contract maybe not deployed.
    if (State.ParliamentContract.Value != null)
    {
      defaultAuthority.OwnerAddress =               State.ParliamentContract.
↪GetDefaultOrganizationAddress.Call(new Empty());
      defaultAuthority.ContractAddress = State.ParliamentContract.Value;
    }
    State.MethodFeeController.Value = defaultAuthority;
}
```

Of course, the authority of SetMethodFee can also be changed, provided that the transaction to modify the authority is sent from the default address of the Parliamentary contract:

```
public override Empty ChangeMethodFeeController(AuthorityInfo input)
{
    RequiredMethodFeeControllerSet();
    AssertSenderAddressWith(State.MethodFeeController.Value.OwnerAddress);
    var organizationExist = CheckOrganizationExist(input);
    Assert(organizationExist, "Invalid authority input.");
    State.MethodFeeController.Value = input;
    return new Empty();
}
```

The implementation of GetMethodFeeController is also very easy

```
public override AuthorityInfo GetMethodFeeController(Empty input)
{
    RequiredMethodFeeControllerSet();
    return State.MethodFeeController.Value;
}
```

Above all, these are the two ways to implement acs1. Mostly, implementations will use a mixture of the two: part of methods' fee is set to a fixed value, the other part of method is not set method fee.

3.1.4 Test

Create ACS1's Stub, and call GetMethodFee and GetMethodFeeController to check if the return value is expected.

3.1.5 Example

All AElf system contracts implement ACS1, which can be used as a reference.

3.2 ACS2 - Parallel Execution Standard

ACS2 is used to provide information for parallel execution of transactions.

3.2.1 Interface

A contract that inherits ACS2 only needs to implement one method:

- GetResourceInfo

The parameter is the Transaction type, and the return value is the type ResourceInfo defined in acs2.proto:

```
message ResourceInfo {
    repeated aelf.ScopedStatePath paths = 1;
    bool non_parallelizable = 2;
}
```

aelf.ScopedStatePath is defined in aelf\core.proto:

```
message ScopedStatePath {
    Address address = 1;
    StatePath path = 2;
}
message StatePath {
    repeated string parts = 1;
}
```

3.2.2 Usage

AElf uses the key-value database to store data. For the data generated during the contract execution, a mechanism called State Path is used to determine the key of the data.

For example Token contract's State file defines a property MappedState < Address, string, long >Balances, it can be used to access, modify balance.

Assuming that the address of the Token contract is Nmjj7noTpMqZ522j76SDsFLhiKkThv1u3d4TxqJMD8v89tWmE. If you want to know the balance of the address 2EM5uV6bSJh6xJfZTUa1pZpYsYcCUAdPvZvFUJzMDJEx3rbioz, you can directly use this key to access redis / ssdb to get its value.

```
Nmjj7noTpMqZ522j76SDsFLhiKkThv1u3d4TxqJMD8v89tWmE/Balances/
↪2EM5uV6bSJh6xJfZTUa1pZpYsYcCUAdPvZvFUJzMDJEx3rbioz/ELF
```

On AElf, the implementation of parallel transaction execution is also based on the key , developers need to provide a method may access to the StatePath, then the corresponding transactions will be properly grouped before executing: if the two methods do not access the same StatePath, then you can safely place them in different groups.

Attention: The transaction will be canceled and labeled to "can not be groupped" when the StatePath mismatchs the method.

If you are interested in the logic, you can view the code ITransactionGrouper, as well as IParallelTransactionExecutingService .

3.2.3 Implementation

A example: within the Token contract, the core logic of method Transfer is to modify the balance of address. It accesses the balances property mentioned above twice.

At this point, we need to notify ITransactionGrouper via the GetResourceInfo method of the key of the ELF balance of address A and address B:

```
var args = TransferInput.Parser.ParseFrom(txn.Params);
var resourceInfo = new ResourceInfo
{
    Paths =
    {
        GetPath(nameof(TokenContractState.Balances), txn.From.ToString(), args.Symbol),
        GetPath(nameof(TokenContractState.Balances), args.To.ToString(), args.Symbol),
    }
};
return resourceInfo;
```

The GetPath forms a ScopedStatePath from several pieces of data that make up the key:

```
private ScopedStatePath GetPath(params string[] parts)
{
    return new ScopedStatePath
    {
        Address = Context.Self,
        Path = new StatePath
        {
            Parts =
            {
                parts
            }
        }
    }
}
```

3.2.4 Test

You can construct two transactions, and the transactions are passed directly to an implementation instance of ITransactionGrouper, and the GroupAsync method is used to see if the two transactions are parallel.

We prepare two stubs that implement the ACS2 contract with different addresses to simulate the Transfer:

```
var keyPair1 = SampleECKeyPairs.KeyPairs[0];
var acs2DemoContractStub1 = GetACS2DemoContractStub(keyPair1);
var keyPair2 = SampleECKeyPairs.KeyPairs[1];
var acs2DemoContractStub2 = GetACS2DemoContractStub(keyPair2);
```

Then take out some services and data needed for testing from Application:

```
var transactionGrouper = Application.ServiceProvider.GetRequiredService
↪<ITransactionGrouper>();
var blockchainService = Application.ServiceProvider.GetRequiredService
↪<IBlockchainService>();
var chain = await blockchainService.GetChainAsync();
```

Finally, check it via transactionGrouper:

```
// Situation can be parallel executed.
{
```

(continues on next page)

(continued from previous page)

```
    var groupedTransactions = await transactionGrouper.GroupAsync(new ChainContext
    {
        BlockHash = chain.BestChainHash,
        BlockHeight = chain.BestChainHeight
    }, new List<Transaction>
    {
        acs2DemoContractStub1.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
        acs2DemoContractStub2.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[3].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
    });
    groupedTransactions.Parallelizables.Count.ShouldBe(2);
}
// Situation cannot.
{
    var groupedTransactions = await transactionGrouper.GroupAsync(new ChainContext
    {
        BlockHash = chain.BestChainHash,
        BlockHeight = chain.BestChainHeight
    }, new List<Transaction>
    {
        acs2DemoContractStub1.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
        acs2DemoContractStub2.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
    });
    groupedTransactions.Parallelizables.Count.ShouldBe(1);
}
```

3.2.5 Example

You can refer to the implementation of the MultiToken contract for GetResourceInfo. Noting that for the ResourceInfo provided by the method Tranfer, you need to consider charging a transaction fee in addition to the two keys mentioned in this article.

3.3 ACS3 - Contract Proposal Standard

Using the AuthorityInfo defined in authority_info.proto restricts a method to be called by a certain address:

```
Assert(Context.Sender == State.AuthorityInfo.Value.OwnerAddress, "No permission.");
```

When a method needs to be agreed by multiple parties, the above solution is obviously inadequate. At this time, you can consider using some of the interfaces provided by ACS3.

3.3.1 Interface

If you want multiple addresses vote to get agreement to do something, you can implement the following methods defined in ACS3:

- CreateProposal, it is to specify a method for a contract and its parameters. When the proposal is approved by multiple addresses, it can be released: use a virtual address as a Sender, and execute this method by sending an inline transaction. Therefore, the parameter CreateProposalInput defines the basic information of the inline transaction to be executed finally. The return value is a hash, which is used to uniquely identify this proposal;
- Approve, Reject, Abstain, the parameters are Hash, called the proposal Id, created by CreateProposal, is used to agree, reject, and abstain respectively .
- Release, the parameter is the proposal Id, is used to release the proposal: when the requirements are met, it can be released;
- ClearProposal is used to clean invalid data in DB.

It can be seen that before a proposal is released, the account with voting rights can agree, object, and abstain. Which specific accounts have the right to vote? ACS3 introduces the concept of Organization. A proposal is attached to an organization from its creation, and only members of the organization can vote.

However, due to the different forms of organization, the Organization structure needs to be defined by the contract implementing the ACS3. Here is an example:

```
message Organization {
    acs3.ProposalReleaseThreshold proposal_release_threshold = 1;
    string token_symbol = 2;
    aelf.Address organization_address = 3;
    aelf.Hash organization_hash = 4;
    acs3.ProposerWhiteList proposer_white_list = 5;
}
```

Because each organization has a default virtual address, adding the code like the begining at this document can verify if the sender is authorized.

```
Assert(Context.Sender == someOrganization.OrganizationAddress, "No permission.");
```

How to know what an orgnanization has agreed on a proposal? ACS3 defines a data structure ProposalReleaseThreshold:

```
message ProposalReleaseThreshold {
    int64 minimal_approval_threshold = 1;
    int64 maximal_rejection_threshold = 2;
    int64 maximal_abstention_threshold = 3;
```

(continues on next page)

(continued from previous page)

```
    int64 minimal_vote_threshold = 4;
}
```

The orgnaization determines how to deal with the proposal according to the data:

- the minimal approval the proposal can be released.
- The most rejection amount the proposal can tolerate.
- The most abstention amount the proposal can tolerate.
- the minimal vote amount the proposal is valid.

Interfaces referencing organization in ACS3:

- ChangeOrganizationThreshold its paramenter is ProposalReleaseThreshold that is used to modify the threshold. Of course, this method also needs permission control
- ChangeOrganizationProposerWhiteList, The organization can restrict which addresses can create proposals. Its parameter is ProposerWhiteList, defined in acs3.proto, which is actually an Address list;
- CreateProposalBySystemContract, The original intention is that the system contract can create a proposal via the virtual address, that is, there are some senders have privileges, and must be a contract:

The type of APIs mentioned above is action, there are some APIs with type View used to query:

- GetProposal is used to get the proposal detailed information.
- ValidateOrganizationExist is used to check if the organization exists in a contract.
- ValidateProposerInWhiteList is used to check if the address is in the whitelist of a organization.

3.3.2 Implementation

It is assumed here that there is only one organization in a contract, that is, there is no need to specifically define the Organization type. Since the organization is not explicitly declared and created, the organization's proposal whitelist does not exist. The process here is that the voter must use a certain token to vote.

For simplicity, only the core methods CreateProposal, Approve, Reject, Abstain, and Release are implemented here.

There are only two necessary State attributes:

```
public MappedState<Hash, ProposalInfo> Proposals { get; set; }
public SingletonState<ProposalReleaseThreshold> ProposalReleaseThreshold { get; set; }
```

The Proposals stores all proposal's information, and the ProposalReleaseThreshold is used to save the requirements that the contract needs to meet to release the proposal.

When the contract is initialized, the proposal release requirements should be set:

```
public override Empty Initialize(Empty input)
{
    State.TokenContract.Value =
        Context.GetContractAddressByName(SmartContractConstants.TokenContractSystemName);
    State.ProposalReleaseThreshold.Value = new ProposalReleaseThreshold
    {
        MinimalApprovalThreshold = 1,
        MinimalVoteThreshold = 1
```

(continues on next page)

(continued from previous page)

```
    };
    return new Empty();
}
```

The requirement is at least one member who vote and at least one approval. Create proposal:

```
public override Hash CreateProposal(CreateProposalInput input)
{
    var proposalId = Context.GenerateId(Context.Self, input.Token);
    Assert(State.Proposals[proposalId] == null, "Proposal with same token already exists.
↪");
    State.Proposals[proposalId] = new ProposalInfo
    {
        ProposalId = proposalId,
        Proposer = Context.Sender,
        ContractMethodName = input.ContractMethodName,
        Params = input.Params,
        ExpiredTime = input.ExpiredTime,
        ToAddress = input.ToAddress,
        ProposalDescriptionUrl = input.ProposalDescriptionUrl
    };
    return proposalId;
}
```

Vote:

```
public override Empty Abstain(Hash input)
{
    Charge();
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    proposal.Abstentions.Add(Context.Sender);
    State.Proposals[input] = proposal;
    return new Empty();
}
public override Empty Approve(Hash input)
{
    Charge();
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    proposal.Approvals.Add(Context.Sender);
    State.Proposals[input] = proposal;
    return new Empty();
}
public override Empty Reject(Hash input)
{
```

(continues on next page)

(continued from previous page)

```
    Charge();
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    proposal.Rejections.Add(Context.Sender);
    State.Proposals[input] = proposal;
    return new Empty();
}
private void Charge()
{
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        To = Context.Self,
        Symbol = Context.Variables.NativeSymbol,
        Amount = 1_00000000
    });
}
```

Release is just count the vote, here is a recommended implementation:

```
public override Empty Release(Hash input)
{
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    Assert(IsReleaseThresholdReached(proposal), "Didn't reach release threshold.");
    Context.SendInline(proposal.ToAddress, proposal.ContractMethodName, proposal.Params);
    return new Empty();
}
private bool IsReleaseThresholdReached(ProposalInfo proposal)
{
    var isRejected = IsProposalRejected(proposal);
    if (isRejected)
        return false;
    var isAbstained = IsProposalAbstained(proposal);
    return !isAbstained && CheckEnoughVoteAndApprovals(proposal);
}
private bool IsProposalRejected(ProposalInfo proposal)
{
    var rejectionMemberCount = proposal.Rejections.Count;
    return rejectionMemberCount > State.ProposalReleaseThreshold.Value.
↪MaximalRejectionThreshold;
}
private bool IsProposalAbstained(ProposalInfo proposal)
{
    var abstentionMemberCount = proposal.Abstentions.Count;
    return abstentionMemberCount > State.ProposalReleaseThreshold.Value.
↪MaximalAbstentionThreshold;
```

(continues on next page)

(continued from previous page)

```
}
private bool CheckEnoughVoteAndApprovals(ProposalInfo proposal)
{
    var approvedMemberCount = proposal.Approvals.Count;
    var isApprovalEnough =
        approvedMemberCount >= State.ProposalReleaseThreshold.Value.
↪MinimalApprovalThreshold;
    if (!isApprovalEnough)
        return false;
    var isVoteThresholdReached =
        proposal.Abstentions.Concat(proposal.Approvals).Concat(proposal.Rejections).
↪Count() >=
        State.ProposalReleaseThreshold.Value.MinimalVoteThreshold;
    return isVoteThresholdReached;
}
```

3.3.3 Test

Before testing, two methods were added to the contract, that had just implemented ACS3. We will test the proposal with these mehods.

Define a singleton string in the State file:

```
public StringState Slogan { get; set; }
```

Then implement a pair of Set/Get methods:

```
public override StringValue GetSlogan(Empty input)
{
    return State.Slogan.Value == null ? new StringValue() : new StringValue {Value =␣
↪State.Slogan.Value};
}
public override Empty SetSlogan(StringValue input)
{
    Assert(Context.Sender == Context.Self, "No permission.");
    State.Slogan.Value = input.Value;
    return new Empty();
}
```

In this way, during the test, create a proposal for the SetSlogan. After passing and releasing, use the GetSlogan method to check whether the Slogan has been modified.

Prepare a Stub that implements the ACS3 contract:

```
var keyPair = SampleECKeyPairs.KeyPairs[0];
var acs3DemoContractStub =
    GetTester<ACS3DemoContractContainer.ACS3DemoContractStub>(DAppContractAddress,␣
↪keyPair);
```

Since approval requires the contract to charge users, the user should send Approve transaction of the Token contract.

```
var tokenContractStub =
    GetTester<TokenContractContainer.TokenContractStub>(
        GetAddress(TokenSmartContractAddressNameProvider.StringName), keyPair);
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = DAppContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
```

Create a proposal, the target method is SetSlogan, here we want to change the Slogan to "AElf" :

```
var proposalId = (await acs3DemoContractStub.CreateProposal.SendAsync(new␣
↪CreateProposalInput
{
    ContractMethodName = nameof(acs3DemoContractStub.SetSlogan),
    ToAddress = DAppContractAddress,
    ExpiredTime = TimestampHelper.GetUtcNow().AddHours(1),
    Params = new StringValue {Value = "AElf"}.ToByteString(),
    Token = HashHelper.ComputeFrom("AElf")
})).Output;
```

Make sure that the Slogan is still an empty string at this time and then vote:

```
// Check slogan
{
    var slogan = await acs3DemoContractStub.GetSlogan.CallAsync(new Empty());
    slogan.Value.ShouldBeEmpty();
}
await acs3DemoContractStub.Approve.SendAsync(proposalId);
```

Release proposal, and the Slogan becomes "AElf".

```
await acs3DemoContractStub.Release.SendAsync(proposalId);
// Check slogan
{
    var slogan = await acs3DemoContractStub.GetSlogan.CallAsync(new Empty());
    slogan.Value.ShouldBe("AElf");
}
```

3.4 ACS4 - Consensus Standard

ACS4 is used to customize consensus mechanisms.

3.4.1 Interface

If you want to customize the consensus mechanism, you need to implement the following five interfaces:

- GetConsensusCommand, whose parameter is a binary array, returns ConsensusCommand defined in acs4.proto. This type is used to indicate the start time of the next block, the block time limit, and the final cut-off time for the account calling GetConsensus Command;

- GetConsensusExtraData, the parameters and return values are binary arrays, which are used to generate consensus block header information through consensus contracts when a new block is produced;
- GenerateConsensusTransactions, the parameter is a binary array, and the return value is of type TransactionList. It is used to generate a consensus system transaction when a block is generated. Each block will contain only one consensus transaction, which is used to write the latest consensus information to the State database;
- ValidateConsensusBeforeExecution, the parameter is a binary array, and the return value is of type ValidationResult, is used to verify whether the consensus information in the block header is correct before the block executes;
- ValidateConsensusAfterExecution, with the same parameter and return value, is used to verify that the consensus information written to the State is correct after the block executes.

ConsensusCommand, ValidationResult and TransactionList are defined as:

```
message ConsensusCommand {
    int32 limit_milliseconds_of_mining_block = 1;// Time limit of mining next block.
    bytes hint = 2;// Context of Hint is diverse according to the consensus protocol we␣
↪choose, so we use bytes.
    google.protobuf.Timestamp arranged_mining_time = 3;
    google.protobuf.Timestamp mining_due_time = 4;
}
message ValidationResult {
    bool success = 1;
    string message = 2;
    bool is_re_trigger = 3;
}
message TransactionList {
    repeated aelf.Transaction transactions = 1;
}
```

3.4.2 Usage

The five interfaces defined in ACS4 basically correspond to the five methods of the IConsensusService interface in the AElf.Kernel.Consensus project:

ACS4	IConsensusService	Methodology	The Timing To Call
`GetConsensusCommand`	Task TriggerConsensusAsync (ChainContext chainContext);	When TriggerConsensusAsync is called, it will use the account configured by the node to call the GetConsensusCommand method of the consensus contract to obtain block information ConsensusCommand), and use it to (see IConsensusScheduler implementation) .	1. When the node is started; 2. When the BestChainFoundEventData event is thrown; 3. When the validation of consensus data fails and the consensus needs to be triggered again (The IsReTrigger field of the ValidationResult type is true);
`GetConsensusExtraData`	Task<byte[]> GetConsensusExtraDataAsync(ChainContext chainContext);	When a node produces a block, it will generate block header information for the new block by IBlockExtraDataService. This service is implemented to traverse all IBlockExtraDataProvider implementations, and they generate binary array information into the ExtraData field of BlockHeader. The consensus block header information is provided by ConsensusExtraDataProvider, in which the GetConsensusExtraDataAsync of the IConsensusService in the consensus contract is called, and the GetConsensusExtraDataAsync method is implemented by calling the GetConsensusExtraData in the consensus contract.	At the time that the node produces a new block.
`GenerateConsensusTransactions`	Task<List<Transaction>> GenerateConsensusTransactionsAsync(ChainContext chainContext);	In the process of generating new blocks, a consensus transaction needs to be generated as one of the system transactions. The basic principle is the same as GetConsensu-	At the time that the node produces a new block.

3.4.3 example

You can refer to the implementation of the AEDPoS contract.

3.5 ACS5 - Contract Threshold Standard

If you want to raise the threshold for using contract, consider implementing ACS5.

3.5.1 Interface

To limit to call a method in a contract, you only need to implement an interface:

- GetMethodCallingThreshold, the parameter is string, and the return value is the MethodCallingThreshold defined in the acs5.proto file.

If you want to modify the threshold after the contract is deployed, another interface can be implemented:

- SetMethodCallingThreshold, the parameter is SetMethodCallingThresholdInput.

The definition of MethodCallingThreshold type is:

```
message MethodCallingThreshold {
    map<string, int64> symbol_to_amount = 1;// The order matters.
    ThresholdCheckType threshold_check_type = 2;
}
enum ThresholdCheckType {
    BALANCE = 0;
    ALLOWANCE = 1;
}
```

The significance of the enumeration ThresholdCheckType is that there are two types of thresholds for contract method calls:

1. It can be called when the balance of a certain token in the account is sufficient, which corresponds to ThresholdCheckType.Balance;
2. Not only does the balance of a token in the account be required to be sufficient, but the account also needs sufficient authorization for the target contract, which corresponds to The ThresholdCheckType.Allowance.
3. SetMethodCallingThresholdInput definition

```
message SetMethodCallingThresholdInput {
    string method = 1;
    map<string, int64> symbol_to_amount = 2;// The order matters.
    ThresholdCheckType threshold_check_type = 3;
}
```

3.5.2 Usage

Similar to ACS1, which uses an automatically generated pre-plugin transaction called ChargeTransactionFees to charge a transaction fee, ACS5 automatically generates a pre-plugin transaction called CheckThreshold to test whether the account that sent the transaction can invoke the corresponding method.

The implementation of CheckThreshold:

```
public override Empty CheckThreshold(CheckThresholdInput input)
{
    var meetThreshold = false;
    var meetBalanceSymbolList = new List<string>();
    foreach (var symbolToThreshold in input.SymbolToThreshold)
    {
        if (GetBalance(input.Sender, symbolToThreshold.Key) < symbolToThreshold.Value)
            continue;
        meetBalanceSymbolList.Add(symbolToThreshold.Key);
    }
    if (meetBalanceSymbolList.Count > 0)
    {
        if (input.IsCheckAllowance)
        {
            foreach (var symbol in meetBalanceSymbolList)
            {
                if (State.Allowances[input.Sender][Context.Sender][symbol] <
                    input.SymbolToThreshold[symbol]) continue;
                meetThreshold = true;
                break;
            }
        }
        else
        {
            meetThreshold = true;
        }
    }
    if (input.SymbolToThreshold.Count == 0)
    {
        meetThreshold = true;
    }
    Assert(meetThreshold, "Cannot meet the calling threshold.");
    return new Empty();
}
```

In other words, if the token balance of the sender of the transaction or the amount authorized for the target contract does not reach the set limit, the pre-plugin transaction will throw an exception, thereby it prevents the original transaction from executing.

3.5.3 Implementation

As the GetMethodFee of ACS1, you can implement only one GetMethodCallingThreshold method.

It can also be achieved by using MappedState<string, MethodCallingThreshold> in the State file:

```
public MappedState<string, MethodCallingThreshold> MethodCallingThresholds { get; set; }
```

But at the same time, do not forget to configure the call permission of SetMethodCallingThreshold, which requires the definition of an Admin in the State (of course, you can also use ACS3):

```
public SingletonState<Address> Admin { get; set; }
```

The easiest implementation

```
public override Empty SetMethodCallingThreshold(SetMethodCallingThresholdInput input)
{
    Assert(State.Admin.Value == Context.Sender, "No permission.");
    State.MethodCallingThresholds[input.Method] = new MethodCallingThreshold
    {
        SymbolToAmount = {input.SymbolToAmount}
    };
    return new Empty();
}
public override MethodCallingThreshold GetMethodCallingThreshold(StringValue input)
{
    return State.MethodCallingThresholds[input.Value];
}
public override Empty Foo(Empty input)
{
    return new Empty();
}
```

3.5.4 Test

You can test the Foo method defined above.

Make a Stub:

```
var keyPair = SampleECKeyPairs.KeyPairs[0];
var acs5DemoContractStub =
    GetTester<ACS5DemoContractContainer.ACS5DemoContractStub>(DAppContractAddress,␣
↪keyPair);
```

Before setting the threshold, check the current threshold, which should be 0:

```
var methodResult = await acs5DemoContractStub.GetMethodCallingThreshold.CallAsync(
    new StringValue
    {
        Value = nameof(acs5DemoContractStub.Foo)
    });
methodResult.SymbolToAmount.Count.ShouldBe(0);
```

The ELF balance of the caller of Foo should be greater than 1 ELF:

```
await acs5DemoContractStub.SetMethodCallingThreshold.SendAsync(
    new SetMethodCallingThresholdInput
    {
        Method = nameof(acs5DemoContractStub.Foo),
        SymbolToAmount =
        {
            {"ELF", 1_0000_0000}
        },
        ThresholdCheckType = ThresholdCheckType.Balance
    });
```

Check the threshold again:

```
methodResult = await acs5DemoContractStub.GetMethodCallingThreshold.CallAsync(
    new StringValue
    {
        Value = nameof(acs5DemoContractStub.Foo)
    });
methodResult.SymbolToAmount.Count.ShouldBe(1);
methodResult.ThresholdCheckType.ShouldBe(ThresholdCheckType.Balance);
```

Send the Foo transaction via an account who has sufficient balance can succeed:

```
// Call with enough balance.
{
    var executionResult = await acs5DemoContractStub.Foo.SendAsync(new Empty());
    executionResult.TransactionResult.Status.ShouldBe(TransactionResultStatus.Mined);
}
```

Send the Foo transaction via another account without ELF fails:

```
// Call without enough balance.
{
    var poorStub =
        GetTester<ACS5DemoContractContainer.ACS5DemoContractStub>(DAppContractAddress,
            SampleECKeyPairs.KeyPairs[1]);
    var executionResult = await poorStub.Foo.SendWithExceptionAsync(new Empty());
    executionResult.TransactionResult.Error.ShouldContain("Cannot meet the calling␣
↪threshold.");
}
```

3.6 ACS8 - Transaction Resource Fee Standard

ACS8 has some similarities to ACS1, both of them are charge transaction fee standard.

The difference is that ACS1 charges the user a transaction fee, ACS8 charges the called contract, and the transaction fee charged by ACS8 is the specified four tokens: WRITE, READ, NET, TRAFFIC.

In another word, if a contract declares that it inherits from ACS8, each transaction in this contract will charge four kinds of resource token.

3.6.1 Interface

Only one method is defined in the acs8.proto file:

- BuyResourceToken, the parameter is the BuyResourceTokenInput defined in the Proto file.

```
message BuyResourceTokenInput {
    string symbol = 1;
    int64 amount = 2;
    int64 pay_limit = 3; // No buy if paying more than this, 0 if no limit
}
```

This method can be used to purchase one of the four resource coins, which consumes the ELF balance in the contract account (you can recharge it yourself, or you can collect the user's ELF tokens as a profit to be self-sufficient).

Of course, it is possible for developers to purchase resource token and then directly transfer to the address of this contract via the Transfer method of the Token contract. Therefore, this interface does not have to be implemented.

3.6.2 Usage

The contract inherited from ACS1 uses a pre-plugin transaction called ChargeTransactionFees for charging transaction fee.

Because the specific charge amount is determined by the actual consumption of the transaction, the post-plugin generates ChargeResourceToken tansaction to charge resource token.

The implementation of ChargeResourceToken is also similar to it of ChargeTransactionFees:

```
public override Empty ChargeResourceToken(ChargeResourceTokenInput input)
{
    Context.LogDebug(() => string.Format("Start executing ChargeResourceToken.{0}",␣
↪input));
    if (input.Equals(new ChargeResourceTokenInput()))
    {
        return new Empty();
    }
    var bill = new TransactionFeeBill();
    foreach (var pair in input.CostDic)
    {
        Context.LogDebug(() => string.Format("Charging {0} {1} tokens.", pair.Value,␣
↪pair.Key));
        var existingBalance = GetBalance(Context.Sender, pair.Key);
        Assert(existingBalance >= pair.Value,
            string.Format("Insufficient resource of {0}. Need balance: {1}; Current␣
↪balance: {2}.", pair.Key, pair.Value, existingBalance));
        bill.FeesMap.Add(pair.Key, pair.Value);
    }
    foreach (var pair in bill.FeesMap)
    {
        Context.Fire(new ResourceTokenCharged
        {
            Symbol = pair.Key,
            Amount = pair.Value,
            ContractAddress = Context.Sender
        });
        if (pair.Value == 0)
        {
            Context.LogDebug(() => string.Format("Maybe incorrect charged resource fee␣
↪of {0}: it's 0.", pair.Key));
        }
    }
    return new Empty();
}
```

The amount of each resource token should be calculated by AElf.Kernel.FeeCalculation. In detail, A data structure named CalculateFeeCoefficients is defined in token_contract.proto, whose function is to save all coefficients of a polynomial, and every three coefficients are a group, such as a, b, c, which means (b / c) * x ^ a. Each resource token has a polynomial that calculates it. Then according to the polynomial and the

actual consumption of the resource, calculate the cost of the resource token. Finally, the cost is used as the parameter of ChargeResourceToken to generate this post-plugin transaction.

In addition, the method of the contract that has been owed cannot be executed before the contract top up resource token. As a result, a pre-plugin transaction is added, similar to the ACS5 pre-plugin transaction, which checks the contract's resource token balance, and the transaction's method name is CheckResourceToken :

```
public override Empty CheckResourceToken(Empty input)
{
    foreach (var symbol in Context.Variables.GetStringArray(TokenContractConstants.
↪PayTxFeeSymbolListName))
    {
        var balance = GetBalance(Context.Sender, symbol);
        var owningBalance = State.OwningResourceToken[Context.Sender][symbol];
        Assert(balance > owningBalance,
            string.Format("Contract balance of {0} token is not enough. Owning {1}.",␣
↪symbol, owningBalance));
    }
    return new Empty();
}
```

3.7 ACS9 - Contract profit dividend standard

On the AElf's side chain, the contract needs to declare where its profits are going, and implemente ACS9.

3.7.1 Interface

ACS9 contains an method which does not have to be implemented:

- TakeContractProfits is used for the developer to collect the profits from the contract. and the profits will be distributed in this method. There are also other methods for developers to claim profits. However, these methods needs to be approved before deployment/upgrade.

Two View methods that must be implemented. They are mostly used in the AElf blockchain browser:

- GetProfitConfig, whose return value is the ProfitConfig defined in acs9.proto, includes the profit token symbol list, the token symbol that the user can lock them to claim the profit, and the portion of the profit that will be donated to the dividend pool each time the developer receives the profit. When reviewing the contract code, you should check if the same ProfitConfig data is actually used for distributing the profits.
- GetProfitsAmount, as the name implies, is used to query the profits of the contract so far, with a return value of type ProfitsMap.

ProfitConfig is defined as:

```
message ProfitConfig {
    int32 donation_parts_per_hundred = 1;
    repeated string profits_token_symbol_list = 2;
    string staking_token_symbol = 3;
}
```

The ProfitsMap type is essentially a map from token symbol to the amount:

```
message ProfitsMap {
    map<string, int64> value = 1;
}
```

3.7.2 Implementation

Here we define a contract. The contract creates a token called APP at the time of initialization and uses the TokenHolder contract to create a token holder bonus scheme with the lock token is designated to APP.

The user will be given 10 APP when to sign up.

Users can purchase 1 APP with 1 ELF using method Deposit, and they can redeem the ELF using the method Withdraw.

When the user sends the Use transaction, the APP token is consumed.

Contract initialization is as follows:

```
public override Empty Initialize(InitializeInput input)
{
    State.TokenHolderContract.Value =
        Context.GetContractAddressByName(SmartContractConstants.
↪TokenHolderContractSystemName);
    State.TokenContract.Value =
        Context.GetContractAddressByName(SmartContractConstants.TokenContractSystemName);
    State.DividendPoolContract.Value =
        Context.GetContractAddressByName(input.DividendPoolContractName.Value.
↪ToBase64());
    State.Symbol.Value = input.Symbol == string.Empty ? "APP" : input.Symbol;
    State.ProfitReceiver.Value = input.ProfitReceiver;
    CreateToken(input.ProfitReceiver);
    // To test TokenHolder Contract.
    CreateTokenHolderProfitScheme();
    // To test ACS9 workflow.
    SetProfitConfig();
    State.ProfitReceiver.Value = input.ProfitReceiver;
    return new Empty();
}
private void CreateToken(Address profitReceiver, bool isLockWhiteListIncludingSelf =␣
↪false)
{
    var lockWhiteList = new List<Address>
        {Context.GetContractAddressByName(SmartContractConstants.
↪TokenHolderContractSystemName)};
    if (isLockWhiteListIncludingSelf)
        lockWhiteList.Add(Context.Self);
    State.TokenContract.Create.Send(new CreateInput
    {
        Symbol = State.Symbol.Value,
        TokenName = "DApp Token",
        Decimals = ACS9DemoContractConstants.Decimal,
        Issuer = Context.Self,
        IsBurnable = true,
```

(continues on next page)

(continued from previous page)

```
        IsProfitable = true,
        TotalSupply = ACS9DemoContractConstants.TotalSupply,
        LockWhiteList =
        {
            lockWhiteList
        }
    });
    State.TokenContract.Issue.Send(new IssueInput
    {
        To = profitReceiver,
        Amount = ACS9DemoContractConstants.TotalSupply / 5,
        Symbol = State.Symbol.Value,
        Memo = "Issue token for profit receiver"
    });
}
private void CreateTokenHolderProfitScheme()
{
    State.TokenHolderContract.CreateScheme.Send(new CreateTokenHolderProfitSchemeInput
    {
        Symbol = State.Symbol.Value
    });
}
private void SetProfitConfig()
{
    State.ProfitConfig.Value = new ProfitConfig
    {
        DonationPartsPerHundred = 1,
        StakingTokenSymbol = "APP",
        ProfitsTokenSymbolList = {"ELF"}
    };
}
```

The State.symbol is a singleton of type string, state.Profitconfig is a singleton of type ProfitConfig, and state.profitreceiver is a singleton of type Address.

The user can use the SighUp method to register and get the bonus. Besides, it will create a archive for him:

```
/// <summary>
/// When user sign up, give him 10 APP tokens, then initialize his profile.
/// </summary>
/// <param name="input"></param>
/// <returns></returns>
public override Empty SignUp(Empty input)
{
    Assert(State.Profiles[Context.Sender] == null, "Already registered.");
    var profile = new Profile
    {
        UserAddress = Context.Sender
    };
    State.TokenContract.Issue.Send(new IssueInput
    {
        Symbol = State.Symbol.Value,
```

(continues on next page)

(continued from previous page)

```
        Amount = ACS9DemoContractConstants.ForNewUser,
        To = Context.Sender
    });
    // Update profile.
    profile.Records.Add(new Record
    {
        Type = RecordType.SignUp,
        Timestamp = Context.CurrentBlockTime,
        Description = string.Format("{0} +{1}",State.Symbol.Value,␣
↪ACS9DemoContractConstants.ForNewUser)
    });
    State.Profiles[Context.Sender] = profile;
    return new Empty();
}
```

Recharge and redemption:

```
public override Empty Deposit(DepositInput input)
{
    // User Address -> DApp Contract.
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        To = Context.Self,
        Symbol = "ELF",
        Amount = input.Amount
    });
    State.TokenContract.Issue.Send(new IssueInput
    {
        Symbol = State.Symbol.Value,
        Amount = input.Amount,
        To = Context.Sender
    });
    // Update profile.
    var profile = State.Profiles[Context.Sender];
    profile.Records.Add(new Record
    {
        Type = RecordType.Deposit,
        Timestamp = Context.CurrentBlockTime,
        Description = string.Format("{0} +{1}", State.Symbol.Value, input.Amount)
    });
    State.Profiles[Context.Sender] = profile;
    return new Empty();
}
public override Empty Withdraw(WithdrawInput input)
{
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        To = Context.Self,
        Symbol = State.Symbol.Value,
        Amount = input.Amount
```

(continues on next page)

(continued from previous page)

```
    });
    State.TokenContract.Transfer.Send(new TransferInput
    {
        To = Context.Sender,
        Symbol = input.Symbol,
        Amount = input.Amount
    });
    State.TokenHolderContract.RemoveBeneficiary.Send(new␣
↪RemoveTokenHolderBeneficiaryInput
    {
        Beneficiary = Context.Sender,
        Amount = input.Amount
    });
    // Update profile.
    var profile = State.Profiles[Context.Sender];
    profile.Records.Add(new Record
    {
        Type = RecordType.Withdraw,
        Timestamp = Context.CurrentBlockTime,
        Description = string.Format("{0} -{1}", State.Symbol.Value, input.Amount)
    });
    State.Profiles[Context.Sender] = profile;
    return new Empty();
}
```

In the implementation of Use, 1/3 profits are directly transferred into the token holder dividend scheme:

```
public override Empty Use(Record input)
{
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        To = Context.Self,
        Symbol = State.Symbol.Value,
        Amount = ACS9DemoContractConstants.UseFee
    });
    if (input.Symbol == string.Empty)
        input.Symbol = State.TokenContract.GetPrimaryTokenSymbol.Call(new Empty()).Value;
    var contributeAmount = ACS9DemoContractConstants.UseFee.Div(3);
    State.TokenContract.Approve.Send(new ApproveInput
    {
        Spender = State.TokenHolderContract.Value,
        Symbol = input.Symbol,
        Amount = contributeAmount
    });
    // Contribute 1/3 profits (ELF) to profit scheme.
    State.TokenHolderContract.ContributeProfits.Send(new ContributeProfitsInput
    {
        SchemeManager = Context.Self,
        Amount = contributeAmount,
        Symbol = input.Symbol
    });
```

(continues on next page)

(continued from previous page)

```
    // Update profile.
    var profile = State.Profiles[Context.Sender];
    profile.Records.Add(new Record
    {
        Type = RecordType.Withdraw,
        Timestamp = Context.CurrentBlockTime,
        Description = string.Format("{0} -{1}", State.Symbol.Value,␣
↪ACS9DemoContractConstants.UseFee),
        Symbol = input.Symbol
    });
    State.Profiles[Context.Sender] = profile;
    return new Empty();
}
```

The implementation of this contract has been completed. Next, implement ACS9 to perfect the profit distribution:

```
public override Empty TakeContractProfits(TakeContractProfitsInput input)
{
    var config = State.ProfitConfig.Value;
    // For Side Chain Dividends Pool.
    var amountForSideChainDividendsPool = input.Amount.Mul(config.
↪DonationPartsPerHundred).Div(100);
    State.TokenContract.Approve.Send(new ApproveInput
    {
        Symbol = input.Symbol,
        Amount = amountForSideChainDividendsPool,
        Spender = State.DividendPoolContract.Value
    });
    State.DividendPoolContract.Donate.Send(new DonateInput
    {
        Symbol = input.Symbol,
        Amount = amountForSideChainDividendsPool
    });
    // For receiver.
    var amountForReceiver = input.Amount.Sub(amountForSideChainDividendsPool);
    State.TokenContract.Transfer.Send(new TransferInput
    {
        To = State.ProfitReceiver.Value,
        Amount = amountForReceiver,
        Symbol = input.Symbol
    });
    // For Token Holder Profit Scheme. (To distribute.)
    State.TokenHolderContract.DistributeProfits.Send(new DistributeProfitsInput
    {
        SchemeManager = Context.Self
    });
    return new Empty();
}
public override ProfitConfig GetProfitConfig(Empty input)
{
    return State.ProfitConfig.Value;
```

(continues on next page)

(continued from previous page)

```
}
public override ProfitsMap GetProfitsAmount(Empty input)
{
    var profitsMap = new ProfitsMap();
    foreach (var symbol in State.ProfitConfig.Value.ProfitsTokenSymbolList)
    {
        var balance = State.TokenContract.GetBalance.Call(new GetBalanceInput
        {
            Owner = Context.Self,
            Symbol = symbol
        }).Balance;
        profitsMap.Value[symbol] = balance;
    }
    return profitsMap;
}
```

3.7.3 Test

Since part of the profits from the ACS9 contract transfer to the Token contract and the other transfer to the dividend pool, a TokenHolder Stub and a contract implementing ACS10 Stub are required in the test. Accordingly, the contracts that implements ACS9 or ACS10 need to be deployed. Before the test begins, the contract implementing ACS9 can be initialized by interface IContractInitializationProvider, and sets the dividend pool's name to the other contract's name:

```
public class ACS9DemoContractInitializationProvider : IContractInitializationProvider
{
    public List<InitializeMethod> GetInitializeMethodList(byte[] contractCode)
    {
        return new List<InitializeMethod>
        {
            new InitializeMethod
            {
                MethodName = nameof(ACS9DemoContract.Initialize),
                Params = new InitializeInput
                {
                    ProfitReceiver = Address.FromPublicKey(SampleECKeyPairs.KeyPairs.
↪Skip(3).First().PublicKey),
                    DividendPoolContractName = ACS10DemoSmartContractNameProvider.Name
                }.ToByteString()
            }
        };
    }
    public Hash SystemSmartContractName { get; } = ACS9DemoSmartContractNameProvider.
↪Name;
    public string ContractCodeName { get; } = "AElf.Contracts.ACS9DemoContract";
}
```

Prepare a user account:

```
protected List<ECKeyPair> UserKeyPairs => SampleECKeyPairs.KeyPairs.Skip(2).Take(3).
↪ToList();
```

Prepare some Stubs:

```
var keyPair = UserKeyPairs[0];
var address = Address.FromPublicKey(keyPair.PublicKey);
// Prepare stubs.
var acs9DemoContractStub = GetACS9DemoContractStub(keyPair);
var acs10DemoContractStub = GetACS10DemoContractStub(keyPair);
var userTokenStub =
    GetTester<TokenContractImplContainer.TokenContractImplStub>(TokenContractAddress,␣
↪UserKeyPairs[0]);
var userTokenHolderStub =
    GetTester<TokenHolderContractContainer.TokenHolderContractStub>
↪(TokenHolderContractAddress,
        UserKeyPairs[0]);
```

Then, transfer ELF to the user (TokenContractStub is the Stub of the initial bp who has much ELF) :

```
// Transfer some ELFs to user.
await TokenContractStub.Transfer.SendAsync(new TransferInput
{
    To = address,
    Symbol = "ELF",
    Amount = 1000_00000000
});
```

Have the user call SignUp to check if he/she has got 10 APP tokens:

```
await acs9DemoContractStub.SignUp.SendAsync(new Empty());
// User has 10 APP tokens because of signing up.
(await GetFirstUserBalance("APP")).ShouldBe(10_00000000);
```

Test the recharge method of the contract itself:

```
var elfBalanceBefore = await GetFirstUserBalance("ELF");
// User has to Approve an amount of ELF tokens before deposit to the DApp.
await userTokenStub.Approve.SendAsync(new ApproveInput
{
    Amount = 1000_00000000,
    Spender = ACS9DemoContractAddress,
    Symbol = "ELF"
});
await acs9DemoContractStub.Deposit.SendAsync(new DepositInput
{
    Amount = 100_00000000
});
// Check the change of balance of ELF.
var elfBalanceAfter = await GetFirstUserBalance("ELF");
elfBalanceAfter.ShouldBe(elfBalanceBefore - 100_00000000);
// Now user has 110 APP tokens.
(await GetFirstUserBalance("APP")).ShouldBe(110_00000000);
```

The user locks up 57 APP via the TokenHolder contract in order to obtain profits from the contract:

```
// User lock some APP tokens for getting profits. (APP -57)
await userTokenHolderStub.RegisterForProfits.SendAsync(new RegisterForProfitsInput
{
    SchemeManager = ACS9DemoContractAddress,
    Amount = 57_00000000
});
```

The Use method is invoked 10 times and 0.3 APP is consumed each time, and finally the user have 50 APP left:

```
await userTokenStub.Approve.SendAsync(new ApproveInput
{
    Amount = long.MaxValue,
    Spender = ACS9DemoContractAddress,
    Symbol = "APP"
});
// User uses 10 times of this DApp. (APP -3)
for (var i = 0; i < 10; i++)
{
    await acs9DemoContractStub.Use.SendAsync(new Record());
}
// Now user has 50 APP tokens.
(await GetFirstUserBalance("APP")).ShouldBe(50_00000000);
```

Using the TakeContractProfits method, the developer attempts to withdraw 10 ELF as profits. The 10 ELF will be transferred to the developer in this method:

```
const long baseBalance = 0;
{
    var balance = await TokenContractStub.GetBalance.CallAsync(new GetBalanceInput
    {
        Owner = UserAddresses[1], Symbol = "ELF"
    });
    balance.Balance.ShouldBe(baseBalance);
}
// Profits receiver claim 10 ELF profits.
await acs9DemoContractStub.TakeContractProfits.SendAsync(new TakeContractProfitsInput
{
    Symbol = "ELF",
    Amount = 10_0000_0000
});
// Then profits receiver should have 9.9 ELF tokens.
{
    var balance = await TokenContractStub.GetBalance.CallAsync(new GetBalanceInput
    {
        Owner = UserAddresses[1], Symbol = "ELF"
    });
    balance.Balance.ShouldBe(baseBalance + 9_9000_0000);
}
```

Next check the profit distribution results. The dividend pool should be allocated 0.1 ELF:

```
// And Side Chain Dividends Pool should have 0.1 ELF tokens.
```

(continues on next page)

(continued from previous page)

```
{
    var scheme = await TokenHolderContractStub.GetScheme.
↪CallAsync(ACS10DemoContractAddress);
    var virtualAddress = await ProfitContractStub.GetSchemeAddress.CallAsync(new␣
↪SchemePeriod
    {
        SchemeId = scheme.SchemeId,
        Period = 0
    });
    var balance = await TokenContractStub.GetBalance.CallAsync(new GetBalanceInput
    {
        Owner = virtualAddress,
        Symbol = "ELF"
    });
    balance.Balance.ShouldBe(1000_0000);
}
```

The user receives 1 ELF from the token holder dividend scheme:

```
// Help user to claim profits from token holder profit scheme.
await TokenHolderContractStub.ClaimProfits.SendAsync(new ClaimProfitsInput
{
    Beneficiary = UserAddresses[0],
    SchemeManager = ACS9DemoContractAddress,
});
// Profits should be 1 ELF.
(await GetFirstUserBalance("ELF")).ShouldBe(elfBalanceAfter + 1_0000_0000);
```

Finally, let's test the Withdraw method.

```
// Withdraw
var beforeBalance =
    await userTokenStub.GetBalance.CallAsync(new GetBalanceInput
    {
        Symbol = "APP",
        Owner = UserAddresses[0]
    });
var withDrawResult = await userTokenHolderStub.Withdraw.
↪SendAsync(ACS9DemoContractAddress);
withDrawResult.TransactionResult.Status.ShouldBe(TransactionResultStatus.Mined);
var resultBalance = await userTokenStub.GetBalance.CallAsync(new GetBalanceInput
{
    Symbol = "APP",
    Owner = UserAddresses[0]
});
resultBalance.Balance.ShouldBe(beforeBalance.Balance + 57_00000000);
```

3.8 ACS10 - dividend pool standard

ACS10 is used to construct a dividend pool in the contract.

3.8.1 Interface

To construct a dividend pool, you can implement the following interfaces optionally:

- Donate is used to donate dividend pool, parameters include the token symbol and the amount to be donated to the dividend pool;
- Release is used to release the dividend pool. The parameter is the number of sessions to release dividends. Be careful to set its calling permission.
- SetSymbolList is used to set the token symbols dividend pool supports. The parameter is of type SymbolList.
- GetSymbolList is used to get the token symbols dividend pool supports. The return type is SymbolList.
- GetUndistributdividends is used to obtain tokens' balance that have not been distributed. The return type is Dividends;
- GetDividends, whose return type is also Dividends, is used to obtain additional dividends from the height of a block.

SymbolList is a string list:

```
message SymbolList {
    repeated string value = 1;
}
```

The type of Dividends is a map from token symbol to amount:

```
message Dividends {
    map<string, int64> value = 1;
}
```

3.8.2 Usage

ACS10 only unifies the standard interface of the dividend pool, which does not interact with the AElf chain.

3.8.3 Implementaion

With the Profit contract

A Profit Scheme can be created using the Profit contract's CreateScheme method:

```
State.ProfitContract.Value =
    Context.GetContractAddressByName(SmartContractConstants.ProfitContractSystemName);
var schemeToken = HashHelper.ComputeFrom(Context.Self);
State.ProfitContract.CreateScheme.Send(new CreateSchemeInput
{
    Manager = Context.Self,
    CanRemoveBeneficiaryDirectly = true,
    IsReleaseAllBalanceEveryTimeByDefault = true,
    Token = schemeToken
});
State.ProfitSchemeId.Value = Context.GenerateId(State.ProfitContract.Value, schemeToken);
```

The Context.GenerateId method is a common method used by the AElf to generate Id. We use the address of the Profit contract and the schemeToken provided to the Profit contract to calculate the Id of the scheme, and we set this id to State.ProfitSchemeId (SingletonState<Hash>).

After the establishment of the dividend scheme:

- ContributeProfits method of Profit can be used to implement the method Donate in ACS10.
- The Release in the ACS10 can be implemented using the method DistributeProfits in the Profit contract;
- Methods such as AddBeneficiary and RemoveBeneficiary can be used to manage the recipients and their weight.
- AddSubScheme, RemoveSubScheme and other methods can be used to manage the sub-dividend scheme and its weight;
- The SetSymbolList and GetSymbolList can be implemented by yourself. Just make sure the symbol list you set is used correctly in Donate and Release.
- GetUndistributedDividends returns the balance of the token whose symbol is includeded in symbol list.

With TokenHolder Contract

When initializing the contract, you should create a token holder dividend scheme using the CreateScheme in the TokenHolder contract(Token Holder Profit Scheme

```
State.TokenHolderContract.Value =
    Context.GetContractAddressByName(SmartContractConstants.
↪TokenHolderContractSystemName);
State.TokenHolderContract.CreateScheme.Send(new CreateTokenHolderProfitSchemeInput
{
    Symbol = Context.Variables.NativeSymbol,
    MinimumLockMinutes = input.MinimumLockMinutes
});
return new Empty();
```

In a token holder dividend scheme, a scheme is bound to its creator, so SchemeId is not necessary to compute (in fact, the scheme is created via the Profit contract).

Considering the GetDividends returns the dividend information according to the input height, so each Donate need update dividend information for each height . A Donate can be implemented as:

```
public override Empty Donate(DonateInput input)
{
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        Symbol = input.Symbol,
        Amount = input.Amount,
        To = Context.Self
    });
    State.TokenContract.Approve.Send(new ApproveInput
    {
        Symbol = input.Symbol,
        Amount = input.Amount,
```

(continues on next page)

(continued from previous page)

```
        Spender = State.TokenHolderContract.Value
    });
    State.TokenHolderContract.ContributeProfits.Send(new ContributeProfitsInput
    {
        SchemeManager = Context.Self,
        Symbol = input.Symbol,
        Amount = input.Amount
    });
    Context.Fire(new DonationReceived
    {
        From = Context.Sender,
        Symbol = input.Symbol,
        Amount = input.Amount,
        PoolContract = Context.Self
    });
    var currentReceivedDividends = State.ReceivedDividends[Context.CurrentHeight];
    if (currentReceivedDividends != null && currentReceivedDividends.Value.
↪ContainsKey(input.Symbol))
    {
        currentReceivedDividends.Value[input.Symbol] =
            currentReceivedDividends.Value[input.Symbol].Add(input.Amount);
    }
    else
    {
        currentReceivedDividends = new Dividends
        {
            Value =
            {
                {
                    input.Symbol, input.Amount
                }
            }
        };
    }
    State.ReceivedDividends[Context.CurrentHeight] = currentReceivedDividends;
    Context.LogDebug(() => string.Format("Contributed {0} {1}s to side chain dividends␣
↪pool.", input.Amount, input.Symbol));
    return new Empty();
}
```

The method Release directly sends the TokenHolder's method DistributeProfits transaction:

```
public override Empty Release(ReleaseInput input)
{
    State.TokenHolderContract.DistributeProfits.Send(new DistributeProfitsInput
    {
        SchemeManager = Context.Self
    });
    return new Empty();
}
```

In the TokenHolder contract, the default implementation is to release what token is received, so SetSymbolList does not need to be implemented, and GetSymbolList returns the symbol list recorded in dividend

scheme:

```
public override Empty SetSymbolList(SymbolList input)
{
    Assert(false, "Not support setting symbol list.");
    return new Empty();
}
public override SymbolList GetSymbolList(Empty input)
{
    return new SymbolList
    {
        Value =
        {
            GetDividendPoolScheme().ReceivedTokenSymbols
        }
    };
}
private Scheme GetDividendPoolScheme()
{
    if (State.DividendPoolSchemeId.Value == null)
    {
        var tokenHolderScheme = State.TokenHolderContract.GetScheme.Call(Context.Self);
        State.DividendPoolSchemeId.Value = tokenHolderScheme.SchemeId;
    }
    return Context.Call<Scheme>(
        Context.GetContractAddressByName(SmartContractConstants.
↪ProfitContractSystemName),
        nameof(ProfitContractContainer.ProfitContractReferenceState.GetScheme),
        State.DividendPoolSchemeId.Value);
}
```

The implementation of GetUndistributdividendeds is the same as described in the previous section, and it returns the balance:

```
public override Dividends GetUndistributedDividends(Empty input)
{
    var scheme = GetDividendPoolScheme();
    return new Dividends
    {
        Value =
        {
            scheme.ReceivedTokenSymbols.Select(s => State.TokenContract.GetBalance.
↪Call(new GetBalanceInput
            {
                Owner = scheme.VirtualAddress,
                Symbol = s
            })).ToDictionary(b => b.Symbol, b => b.Balance)
        }
    };
}
```

In addition to the Profit and TokenHolder contracts, of course, you can also implement a dividend pool on your own contract.

3.8.4 Test

The dividend pool, for example, is tested in two ways with the token Holder contract.

One way is for the dividend pool to send Donate, Release and a series of query operations;

The other way is to use an account to lock up, and then take out dividends.

Define the required Stubs:

```
const long amount = 10_00000000;
var keyPair = SampleECKeyPairs.KeyPairs[0];
var address = Address.FromPublicKey(keyPair.PublicKey);
var acs10DemoContractStub =
    GetTester<ACS10DemoContractContainer.ACS10DemoContractStub>(DAppContractAddress,␣
↪keyPair);
var tokenContractStub =
    GetTester<TokenContractContainer.TokenContractStub>(TokenContractAddress, keyPair);
var tokenHolderContractStub =
    GetTester<TokenHolderContractContainer.TokenHolderContractStub>
↪(TokenHolderContractAddress,
        keyPair);
```

Before proceeding, You should Approve the TokenHolder contract and the dividend pool contract.

```
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = TokenHolderContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = DAppContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
```

Lock the position, at which point the account balance is reduced by 10 ELF:

```
await tokenHolderContractStub.RegisterForProfits.SendAsync(new RegisterForProfitsInput
{
    SchemeManager = DAppContractAddress,
    Amount = amount
});
```

Donate, at which point the account balance is reduced by another 10 ELF:

```
await acs10DemoContractStub.Donate.SendAsync(new DonateInput
{
    Symbol = "ELF",
    Amount = amount
});
```

At this point you can test the GetUndistributedDividends and GetDividends:

```
// Check undistributed dividends before releasing.
{
    var undistributedDividends =
        await acs10DemoContractStub.GetUndistributedDividends.CallAsync(new Empty());
    undistributedDividends.Value["ELF"].ShouldBe(amount);
}
var blockchainService = Application.ServiceProvider.GetRequiredService
↪<IBlockchainService>();
var currentBlockHeight = (await blockchainService.GetChainAsync()).BestChainHeight;
var dividends =
    await acs10DemoContractStub.GetDividends.CallAsync(new Int64Value {Value =␣
↪currentBlockHeight});
dividends.Value["ELF"].ShouldBe(amount);
```

Release bonus, and test GetUndistributedDividends again:

```
await acs10DemoContractStub.Release.SendAsync(new ReleaseInput
{
    PeriodNumber = 1
});
// Check undistributed dividends after releasing.
{
    var undistributedDividends =
        await acs10DemoContractStub.GetUndistributedDividends.CallAsync(new Empty());
    undistributedDividends.Value["ELF"].ShouldBe(0);
}
```

Finally, let this account receive the dividend and then observe the change in its balance:

```
var balanceBeforeClaimForProfits = await tokenContractStub.GetBalance.CallAsync(new␣
↪GetBalanceInput
{
    Owner = address,
    Symbol = "ELF"
});
await tokenHolderContractStub.ClaimProfits.SendAsync(new ClaimProfitsInput
{
    SchemeManager = DAppContractAddress,
    Beneficiary = address
});
var balanceAfterClaimForProfits = await tokenContractStub.GetBalance.CallAsync(new␣
↪GetBalanceInput
{
    Owner = address,
    Symbol = "ELF"
});
balanceAfterClaimForProfits.Balance.ShouldBe(balanceBeforeClaimForProfits.Balance +␣
↪amount);
```

3.8.5 Example

The dividend pool of the main chain and the side chain is built by implementing ACS10.

The dividend pool provided by the Treasury contract implementing ACS10 is on the main chain.

The dividend pool provided by the ACS10 contract implementing ACS10 is on the side chain.

CHAPTER 4

Demo

4.1 Bingo Game

4.1.1 Requirement Analysis

Basic Requirement

Only one rule Users can bet a certain amount of ELF on Bingo contract, and then users will gain more ELF or to lose all ELF bet before in the expected time.

For users, operation steps are as follows:

1. Send an Approve transaction by Token Contract to grant Bingo Contract amount of ELF.
2. Bet by Bingo Contract, and the outcome will be unveiled in the expected time.
3. After a certain time, or after the block height is reached, the user can use the Bingo contract to query the results, and at the same time, the Bingo contract will transfer a certain amount of ELF to the user (If the amount at this time is greater than the bet amount, it means that the user won; vice versa).

4.1.2 API List

In summary, two basic APIs are needed:

1. Play, corresponding to step 2;
2. Bingo, corresponding to step 3.

In order to make the Bingo contract a more complete DApp contract, two additional Action methods are added:

1. Register, which creates a file for users, can save the registration time and user's eigenvalues (these eigenvalues participate in the calculation of the random number used in the Bingo game);
2. Quit, which deletes users' file.

In addition, there are some View methods for querying information only:

1. GetAward, which allows users to query the award information of a bet;
2. GetPlayerInformation, used to query player's information.

Method	Parameters	Return	function
`Register`	Empty	Empty	register player information
`Quit`	Empty	Empty	delete player information
`Play`	Int64Value anount you debt	Int64Value the resulting block height	debt
`Bingo`	Hash the transaction id of Play	Empty True indicates win	query the game's result
`GetAward`	Hash the transaction id of Play	Int64Value award	query the amount of award
`GetPlayerInformation`	Address player's address	Player- Information	query player's information

4.1.3 Write Contract

Use the code generator to generate contracts and test projects

Open the AElf.Boilerplate.CodeGenerator project in the AElf.Boilerplate solution, and modify the Contents node in appsetting.json under this project:

```
{
  "Contents": [
  {
    "Origin": "AElf.Contracts.HelloWorldContract",
    "New": "AElf.Contracts.BingoContract"
  },
  {
    "Origin": "HelloWorld",
    "New": "Bingo"
  },
  {
    "Origin": "hello_world",
    "New": "bingo"
  }
  ],
}
```

Then run the AElf.Boilerplate.CodeGenerator project. After running successfully, you will see a AElf.Contracts.BingoContract.sln in the same directory as the AElf.Boilerplate.sln is in. After opening the sln, you will see that the contract project and test case project of the Bingo contract have been generated and are included in the new solution.

Define Proto

Based on the API list in the requirements analysis, the bingo_contract.proto file is as follows:

```
syntax = "proto3";
import "aelf/core.proto";
import "aelf/options.proto";
import "google/protobuf/empty.proto";
import "google/protobuf/wrappers.proto";
import "google/protobuf/timestamp.proto";
option csharp_namespace = "AElf.Contracts.BingoContract";
service BingoContract {
    option (aelf.csharp_state) = "AElf.Contracts.BingoContract.BingoContractState";

    // Actions
    rpc Register (google.protobuf.Empty) returns (google.protobuf.Empty) {
    }
    rpc Play (google.protobuf.Int64Value) returns (google.protobuf.Int64Value) {
    }
    rpc Bingo (aelf.Hash) returns (google.protobuf.BoolValue) {
    }
    rpc Quit (google.protobuf.Empty) returns (google.protobuf.Empty) {
    }

    // Views
    rpc GetAward (aelf.Hash) returns (google.protobuf.Int64Value) {
        option (aelf.is_view) = true;
    }
    rpc GetPlayerInformation (aelf.Address) returns (PlayerInformation) {
        option (aelf.is_view) = true;
    }
}
message PlayerInformation {
    aelf.Hash seed = 1;
    repeated BoutInformation bouts = 2;
    google.protobuf.Timestamp register_time = 3;
}
message BoutInformation {
    int64 play_block_height = 1;
    int64 amount = 2;
    int64 award = 3;
    bool is_complete = 4;
    aelf.Hash play_id = 5;
    int64 bingo_block_height = 6;
}
```

Contract Implementation

Here only talk about the general idea of the Action method, specifically need to turn the code:

https://github.com/AElfProject/aelf-boilerplate/blob/preview-3/chain/contract/AElf.Contracts.BingoGameContract/BingoGameContract.cs

Register & Quit

Register

1. Determine the Seed of the user, Seed is a hash value, participating in the calculation of the random number, each user is different, so as to ensure that different users get different results on the same height;
2. Record the user's registration time.

Quit Just delete the user's information.

Play & Bingo

Play

1. Use TransferFrom to deduct the user's bet amount;
2. At the same time add a round (Bount) for the user, when the Bount is initialized, record three messages 1.PlayId, the transaction Id of this transaction, is used to uniquely identify the Bout (see BoutInformation for its data structure in the Proto definition); 2.Amount Record the amount of the bet 3.Record the height of the block in which the Play transaction is packaged.

Bingo

1. Find the corresponding Bout according to PlayId, if the current block height is greater than PlayBlockHeight + number of nodes * 8, you can get the result that you win or lose;
2. Use the current height and the user's Seed to calculate a random number, and then treat the hash value as a bit Array, each of which is added to get a number ranging from 0 to 256.
3. Whether the number is divisible by 2 determines the user wins or loses;
4. The range of this number determines the amount of win/loss for the user, see the note of GetKind method for details.

4.1.4 Write Test

Because the token transfer is involved in this test, in addition to constructing the stub of the bingo contract, the stub of the token contract is also required, so the code referenced in csproj for the proto file is:

```
<ItemGroup>
  <ContractStub Include="..\..\protobuf\bingo_contract.proto">
    <Link>Protobuf\Proto\bingo_contract.proto</Link>
  </ContractStub>
  <ContractStub Include="..\..\protobuf\token_contract.proto">
    <Link>Protobuf\Proto\token_contract.proto</Link>
  </ContractStub>
</ItemGroup>
```

Then you can write test code directly in the Test method of BingoContractTest. Prepare the two stubs mentioned above:

```
// Get a stub for testing.
var keyPair = SampleECKeyPairs.KeyPairs[0];
var stub = GetBingoContractStub(keyPair);
```

(continues on next page)

(continued from previous page)

```
var tokenStub =
    GetTester<TokenContractContainer.TokenContractStub>(
        GetAddress(TokenSmartContractAddressNameProvider.StringName), keyPair);
```

The stub is the stub of the bingo contract, and the tokenStub is the stub of the token contract.

In the unit test, the keyPair account is given a large amount of ELF by default, and the bingo contract needs a certain bonus pool to run, so first let the account transfer ELF to the bingo contract:

```
// Prepare awards.
await tokenStub.Transfer.SendAsync(new TransferInput
{
    To = DAppContractAddress,
    Symbol = "ELF",
    Amount = 100_00000000
});
```

Then you can start using the Bingo contract. Register

```
await stub.Register.SendAsync(new Empty());
```

After registration, take a look at PlayInformation:

```
// Now I have player information.
var address = Address.FromPublicKey(keyPair.PublicKey);
{
    var playerInformation = await stub.GetPlayerInformation.CallAsync(address);
    playerInformation.Seed.Value.ShouldNotBeEmpty();
    playerInformation.RegisterTime.ShouldNotBeNull();
}
```

Bet, but before you can bet, you need to Approve the bingo contract:

```
// Play.
await tokenStub.Approve.SendAsync(new ApproveInput
{
    Spender = DAppContractAddress,
    Symbol = "ELF",
    Amount = 10000
});
await stub.Play.SendAsync(new Int64Value {Value = 10000});
```

See if Bout is generated after betting.

```
Hash playId;
{
    var playerInformation = await stub.GetPlayerInformation.CallAsync(address);
    playerInformation.Bouts.ShouldNotBeEmpty();
    playId = playerInformation.Bouts.First().PlayId;
}
```

Since the outcome requires eight blocks, you need send seven invalid transactions (these transactions will fail, but the block height will increase) :

```
// Mine 7 more blocks.
for (var i = 0; i < 7; i++)
{
    await stub.Bingo.SendWithExceptionAsync(playId);
}
```

Last check the award, and that the award amount is greater than 0 indicates you win.

```
await stub.Bingo.SendAsync(playId);
var award = await stub.GetAward.CallAsync(playId);
award.Value.ShouldNotBe(0);
```

4.2 todo

4.3 todo

4.4 todo

CHAPTER 5

DAPP

5.1 todo

5.2 todo

5.3 todo