AElf

Release release/1.2.3

Feb 01, 2023

Contents

CHAPTER 1

Welcome to AElf's official documentation.

This site is where we centralize our guides, documents and api references. Wether you're a dApp developer looking to build some awesome apps on top of AElf or simply just interested in seeing what a running node looks like, this place is for you!

As of today the documentation is correct but still a work in progress so we invite you to frequently visit and discover any new content.

CHAPTER 2

Development Environment

2.1 Install

Before you get started with the tutorials, you need to install the following tools and frameworks.

For most of these dependencies, we provide command line instructions for macOS, Linux Ubuntu 18, and Windows. In case any problems occur or if you have more complex needs, please leave a message on GitHub and we will handle it ASAP.

2.1.1 macOS

Configure Environment

You can install and set up the development environment on macOS computers with either Intel or Apple M1 processors. This will take 10-20 minutes.

Before You Start

Before you install and set up the development environment on a macOS device, please make sure that your computer meets these basic requirements:

- Operating system version is 10.7 Lion or higher.
- At least a 2Ghz processor, 3Ghz recommended.
- At least 8 GB RAM, 16 GB recommended.
- No less than 10 GB of available space.
- Broadband internet connection.

Support for Apple M1

If you use a macOS computer with an Apple M1 chip, you need to install Apple Rosetta. Open the Terminal on your computer and execute this command,Please be patient while the command is executed.

```
/usr/sbin/softwareupdate --install-rosetta --agree-to-license
```

Install Homebrew

In most cases, you should use Homebrew to install and manage packages on macOS devices. If Homebrew is not installed on your local computer yet, you should download and install it before you continue.

To install Homebrew:

1. Open Terminal.
2. Execute this command to install Homebrew:

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   ↪HEAD/install.sh)"
   ```

3. Execute this command to check if Homebrew is installed:

   ```
   brew --version
   ```

The following output suggests successful installation:

```
Homebrew 3.3.1

Homebrew/homebrew-core (git revision c6c488fbc0f; last commit 2021-10-30)

Homebrew/homebrew-cask (git revision 66bab33b26; last commit 2021-10-30)
```

Environment Update

Execute this command to update your environment:

```
brew update
```

You will see output like this.

```
You have xx outdated formula installed.
You can upgrade it with brew upgrade
or list it with brew outdated.
```

You can execute the following command to upgrade or skip to the installation of Git.

```
brew upgrade
```

Install Git

If you want to use our customized smart contract development environment or to run a node, you need to clone aelf's repo (download source code). As aelf's code is hosted on GitHub, you need to install **Git** first.

1. Execute this command in Terminal:

```
brew install git
```

2. Execute this command to check if Git is installed:

```
git --version
```

The following output suggests successful installation:

```
git version xx.xx.xx
```

Install .NET SDK

As aelf is mostly developed with .NET Core, you need to download and install .NET Core SDK (Installers - x64 recommended for macOS devices with Intel processors; Installers - Arm64 recommended for macOS devices with M1 chips).

1. Download and install .NET 6.0 which is currently used in aelf's repo.
2. Please reopen Terminal after the installation is done.
3. Execute this command to check if .NET is installed:

```
dotnet --version
```

The following output suggests successful installation:

```
6.0.403
```

Install protoBuf

1. Execute this command to install protoBuf:

```
brew install protobuf
```

If it shows error `Permission denied @ apply2files`, then there is a permission issue. You can solve it using the following command and then redo the installation with the above command:

```
sudo chown -R $(whoami) $(brew --prefix)/*
```

2. Execute this command to check if protoBuf is installed:

```
protoc --version
```

The following output suggests successful installation:

```
libprotoc 3.21.9
```

Install Redis

1. Execute this command to install Redis:

```
brew install redis
```

2. Execute this command to start a Redis instance and check if Redis is installed:

```
redis-server
```

The following output suggests Redis is installed and a Redis instance is started:

Install Nodejs

1. Execute this command to install Nodejs:

```
brew install node
```

2. Execute this command to check if Nodejs is installed:

```
npm --version
```

The following output suggests successful installation:

```
6.14.8
```

2.1.2 Linux

Configure Environment

You can install and set up the development environment on computers running 64-bit Linux. This will take 10-20 minutes.

Before You Start

Before you install and set up the development environment on a Linux device, please make sure that your computer meets these basic requirements:

- Ubuntu 18.
- Broadband internet connection.

Update Environment

Execute this command to update your environment, Please be patient while the command is executed:

```
sudo apt-get update
```

The following output suggests successful update:

```
Fetched 25.0 MB in 3s (8,574 kB/s)
Reading package lists... Done
```

Install Git

If you want to use our customized smart contract development environment or to run a node, you need to clone aelf's repo (download source code). As aelf's code is hosted on GitHub, you need to install **Git** first.

1. Open the terminal.
2. Execute this command to install Git:

   ```
   sudo apt-get install git -y
   ```

3. Execute this command to check if Git is installed:

   ```
   git --version
   ```

The following output suggests successful installation:

```
git version 2.17.1
```

Install .NET SDK

As aelf is mostly developed with .NET Core, you need to download and install .NET Core SDK.

1. Execute the following commands to install .NET 6.0.
   1. Execute this command to download .NET packages:

      ```
      wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-
      ↪prod.deb -O packages-microsoft-prod.deb
      ```

   2. Execute this command to unzip .NET packages:

      ```
      sudo dpkg -i packages-microsoft-prod.deb

      rm packages-microsoft-prod.deb
      ```

3. Execute this command to install .NET:

```
sudo apt-get update && \

sudo apt-get install -y dotnet-sdk-6.0
```

2. Execute this command to check if .NET 6.0 is installed:

```
dotnet --version
```

The following output suggests successful installation:

```
6.0.403
```

Install protoBuf

Before you start the installation, please check the directory you use and execute the following commands to install.

1. Execute the following commands to install protoBuf.

   1. Execute this command to download protoBuf packages:

   ```
   curl -OL https://github.com/google/protobuf/releases/download/v21.9/protoc-21.
   ↪9-linux-x86_64.zip
   ```

   2. Execute this command to unzip protoBuf packages:

   ```
   unzip protoc-21.9-linux-x86_64.zip -d protoc3
   ```

   3. Execute these commands to install protoBuf:

   ```
   sudo mv protoc3/bin/* /usr/local/bin/

   sudo mv protoc3/include/* /usr/local/include/

   sudo chown ${USER} /usr/local/bin/protoc

   sudo chown -R ${USER} /usr/local/include/google
   ```

   If it shows error `Permission denied @ apply2files`, then there is a permission issue. You can solve it using the following command and then redo the installation with the above commands:

   ```
   sudo chown -R $(whoami) $(brew --prefix)/*
   ```

2. Execute this command to check if protoBuf is installed:

```
protoc --version
```

The following output suggests successful installation:

```
libprotoc 3.21.9
```

Install Redis

1. Execute this command to install Redis:

```
sudo apt-get install redis -y
```

2. Execute this command to start a Redis instance and check if Redis is installed:

```
redis-server
```

The following output suggests Redis is installed and a Redis instance is started:

```
Server initialized
Ready to accept connections
```

You can open a new terminal and use redis-cli to start Redis command line. The command below can be used to clear Redis cache (be careful to use it):

```
flushall
```

Install Nodejs

1. Execute these commands to install Nodejs:

```
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -

sudo apt-get install -y nodejs
```

2. Execute this command to check if Nodejs is installed:

```
npm --version
```

The following output suggests successful installation:

```
6.14.8
```

2.1.3 Windows

Configure Environment

You can install and set up the development environment on computers running Windows 10 or higher. This will take 10-20 minutes.

Before You Start

Before you install and set up the development environment on a Windows device, please make sure that your computer meets these basic requirements:

- Operating system version is Windows 10 or higher.
- Broadband internet connection.

Install Chocolatey (Recommended)

Chocolatey is an open-source package manager for Windows software that makes installation simpler, like Homebrew for Linux and macOS. If you don't want to install it, please use the provided download links for each software to complete their installation.

1. Open **cmd** or **PowerShell** as administrator (Press Win + x).
2. Execute the following commands in order and enter y to install Chocolatey, Please be patient while the command is executed:

```
Set-ExecutionPolicy AllSigned

Set-ExecutionPolicy Bypass -Scope Process

Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.
↪WebClient).DownloadString('https://chocolatey.org/install.ps1'))

Set-ExecutionPolicy RemoteSigned
```

3. Execute this command to check if Chocolatey is installed:

```
choco
```

The following output suggests successful installation:

```
Chocolatey vx.x.x
```

If it shows`The term 'choco' is not recognized as the name of a cmdlet, function, script file, or operable program`, then there is a permission issue with PowerShell. To solve it:

- **Right-click** the computer icon and select **Properties**.
- Click **Advanced** in **System Properties** and select **Environment Variables** on the bottom right.
- Check if the **ChocolateyInstall variable** is in **System variables**, and its default value is the Chocolatey installation path `C:\Program Files\Chocolatey`. If you don't find it, click New System Variable to manually add it.

Install Git

If you want to use our customized smart contract development environment or to run a node, you need to clone aelf's repo (download source code). As aelf's code is hosted on GitHub, you need to install **Git** first.

1. You can download Git through this link or execute this command in cmd or PowerShell:

```
choco install git -y
```

2. Execute this command to check if Git is installed:

```
git --version
```

The following output suggests successful installation:

```
git version xx.xx.xx
```

If it shows `The term 'git' is not recognized as the name of a cmdlet, function, script file, or operable program`, you can:

- **Right-click** the computer icon and select **Properties**.
- Click **Advanced** in **System Properties** and select **Environment Variables** on the bottom right.
- Check if the Git variable is in **Path** in **System variables**, and its default value is the Git installation path `C:\Program Files\git`. If you don't find it, click **New System Variable** to manually add it.

Install .NET SDK

As aelf is mostly developed with .NET Core, you need to download and install .NET Core SDK (Installers - x64 recommended for Windows devices).

1. Download and install .NET 6.0 which is currently used in aelf's repo.
2. Please reopen cmd or PowerShell after the installation is done.
3. Execute this command to check if .NET is installed:

   ```
   dotnet --version
   ```

   The following output suggests successful installation:

   ```
   6.0.403
   ```

Install protoBuf

1. You can download protoBuf through this link or execute this command in cmd or PowerShell:

   ```
   choco install protoc --version=3.11.4 -y

   choco install unzip -y
   ```

2. Execute this command to check if protoBuf is installed:

   ```
   protoc --version
   ```

The following output suggests successful installation:

```
libprotoc 3.21.9
```

Install Redis

1. You can download Redis through MicroSoftArchive-Redis or execute this command in cmd or PowerShell:

   ```
   choco install redis-64 -y
   ```

2. Execute this command to start a Redis instance and check if Redis is installed:

   ```
   memurai
   ```

The following output suggests Redis is installed and a Redis instance is started:

Install Nodejs

1. You can download Nodejs through Node.js or execute this command in cmd or PowerShell:

```
choco install nodejs -y
```

2. Execute this command to check if Nodejs is installed:

```
npm --version
```

The following output suggests successful installation:

```
6.14.8
```

If it shows The term 'npm' is not recognized as the name of a cmdlet, function, script file, or operable program, you can:

- **Right-click** the computer icon and select **Properties**.
- Click **Advanced** in **System Properties** and select **Environment Variables** on the bottom right.
- Check if the Nodejs variable is in **Path** in **System variables**, and its default value is the Nodejs installation path `C:\Program Files\nodejs`. If you don't find it, click **New System Variable** to manually add it.

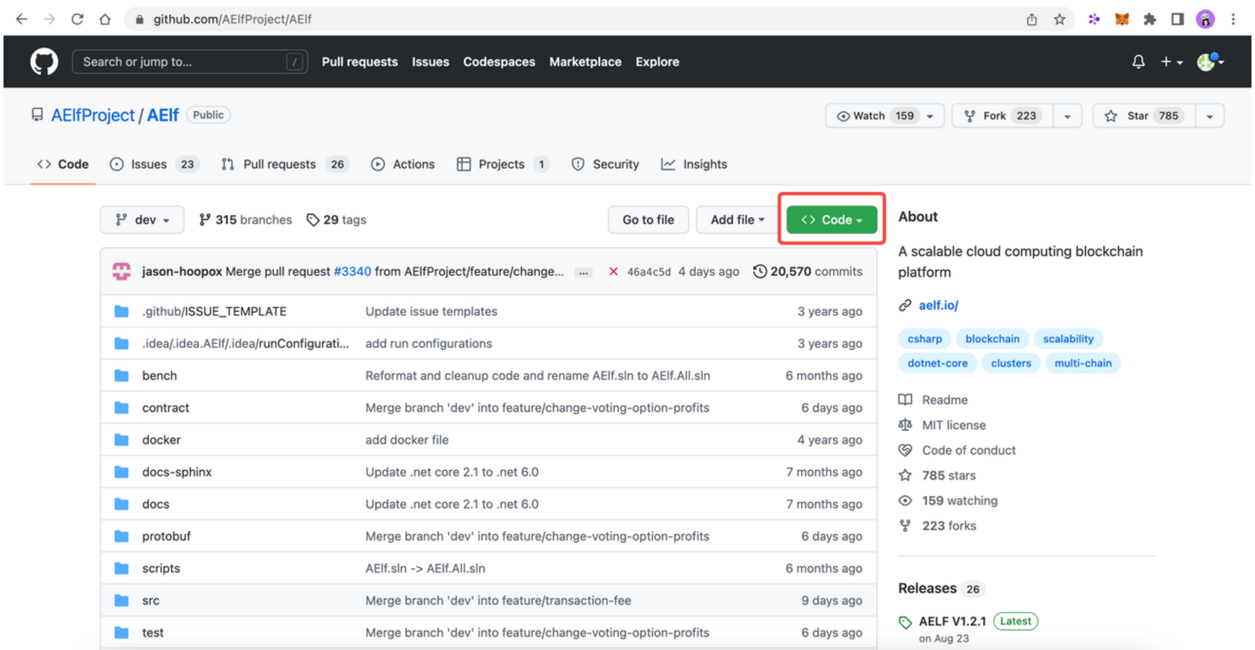
2.1.4 Codespaces

A codespace is an instant development environment that's hosted in the cloud. It provides users with general-purpose programming languages and tooling through containers. You can install and set up the development environment in

Codespaces. This will take 10-20 minutes. Please be patient while the command is executed.

Basic Environment Configurations

1. Visit AElfProject / AElf via a browser.
2. Click the green **Code** button on the top right.



3. Select `Codespaces` and click +.

Then a new tab will be opened that shows the `Codespaces` interface. After the page is loaded, you will see:

- The left side displays all the content in this repo.
- The upper right side is where you can write code or view text.
- The lower right side is a terminal where you can build and run code (If the terminal doesn't open by default, you can click the hamburger menu on the top left and select Terminal -> New Terminal, or press control + shift + ' on your keyboard).

Currently, `Codespaces` have completed the configuration for part of the environments, yet there are some you need to manually configure.

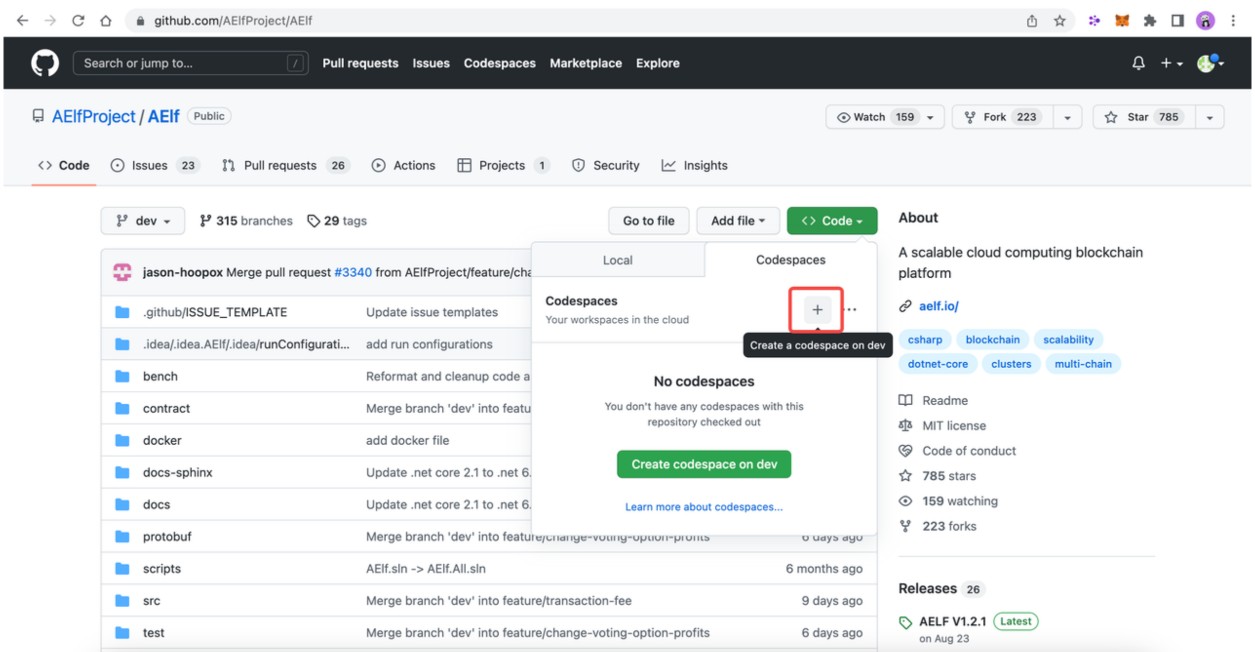
At the time of writing, `Codespaces` have done the configuration for git and nodejs. You can type the following commands to check their versions:

```
# git version 2.25.1
git --version

# 8.19.2
npm --version
```

Update Environment

Execute this command to update your environment:




```
sudo apt-get update
```

The following output suggests successful update:

```
Fetched 25.0 MB in 3s (8,574 kB/s)
Reading package lists... Done
```

Install .NET SDK

.NET SDK 7.0 is used in this repo. Hence, you need to reinstall v6.0 otherwise there will be building issues.

1. Execute this command to check if v7.0 is used:

   ```
   # 7.0.100
   dotnet --version
   ```

   If there is v7.0, execute this command to delete it:

   ```
   sudo rm -rf /home/codespace/.dotnet/*
   ```

2. Execute this command to reinstall v6.0:

   ```
   wget https://packages.microsoft.com/config/ubuntu/22.04/packages-microsoft-prod.
   ↪deb -O packages-microsoft-prod.deb

   sudo dpkg -i packages-microsoft-prod.deb

   rm packages-microsoft-prod.deb

   sudo apt-get update && \

   sudo apt-get install -y dotnet-sdk-6.0
   ```

3. Restart bash after the installation and execute this command to check if v6.0 is installed:

```
# 6.0.403
dotnet --version
```

The following output suggests successful installation:

```
6.0.403
```

Install protoBuf

1. Execute this command to install protoBuf:

```
curl -OL https://github.com/google/protobuf/releases/download/v21.9/protoc-21.9-
↪linux-x86_64.zip
unzip protoc-21.9-linux-x86_64.zip -d protoc3

sudo mv protoc3/bin/* /usr/local/bin/

sudo mv protoc3/include/* /usr/local/include/

sudo chown ${USER} /usr/local/bin/protoc

sudo chown -R ${USER} /usr/local/include/google
```

2. Execute this command to check if protoBuf is installed:

```
protoc --version
```

The following output suggests successful installation:

```
libprotoc 3.21.9
```

Install Redis

1. Execute this command to install Redis:

```
sudo apt-get install redis -y
```

2. Execute this command to start a Redis instance and check if Redis is installed:

```
redis-server
```

The following output suggests Redis is installed and a Redis instance is started:

```
Server initialized
Ready to accept connections
```

What's Next

If you have already installed the tools and frameworks above, you can skip this step. For info about contract deployment and nodes running, please read the following:

Smart contract development

Smart contract deployment

Node

2.2 Node

If you already know something about aelf blockchain and want to get deeply involved, you can proceed with the following and run your own node.

If you are a beginner or you want to deploy contracts onto aelf, please click here to learn more.

Why Should I Run a Node

- Full node: A full node stores the complete blockchain data and you can view all the info. It also enables you to deploy DApps and contracts on aelf or interact with its contracts.
- BP: To run a full node that produces blocks, the node needs to participate in the election. If ranked among the top 2N+1 (N=8 in the first year and increases by 1 every year. Currently the threshold is 17), it can get involved in the governance of aelf.

Next, we will show you how to deploy nodes.

2.2.1 Single Node

macOS

Follow this doc to run an aelf single node on a macOS device and this will take around 20 minutes to complete.

Install aelf-command

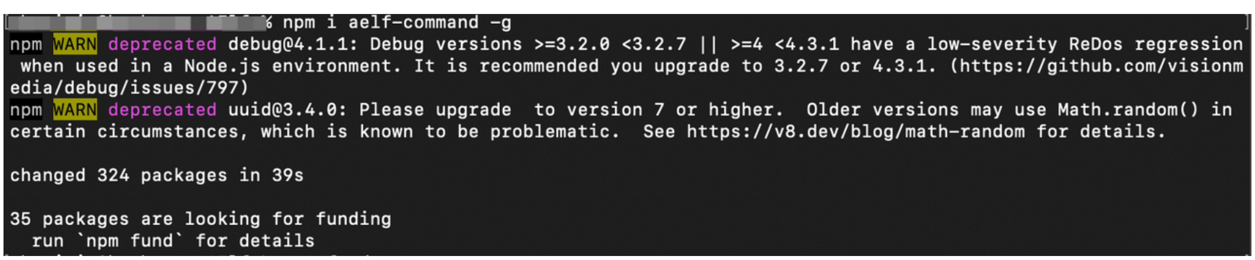
Execute this command to install aelf-command:

```
npm i aelf-command -g
```

The following output suggests successful installation:

```
+ aelf-command@0.1.44
added 314 packages from 208 contributors in 25.958s
```

Besides, you might see warnings like this due to differences in system configuration. Please ignore it.



If it shows error `Permission denied @ apply2files`, then there is a permission issue. You can solve it using the following command and then redo the installation with the above command:

```
sudo chmod 755 /usr/local/lib/node_modules
```

Clone and Build aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command to create a directory:

```
mkdir ~/Desktop/Code
```

2. Execute this command to change the directory:

```
cd ~/Desktop/Code
```

3. Execute this command to clone aelf's code:

```
git clone https://github.com/AElfProject/AElf.git
```

4. Execute this command to change to aelf's directory:

```
cd AElf
```

5. Execute this command to restore aelf's files:

```
dotnet restore AElf.All.sln
```

6. Execute this command to build aelf's code (this will take several minutes):

```
dotnet build AElf.All.sln
```

The following output suggests successful building:

```
 xx Warning(s)
    0 Error(s)

Time Elapsed 00:15:59.77
```

If contract_csharp_plugin fails to be called, it may be because you don't have Rosetta 2 installed. Please execute this command and then retry:

```
/usr/sbin/softwareupdate --install-rosetta --agree-to-license
```

Create an aelf Account

Execute this command:

```
aelf-command create
```

An aelf account will be automatically created and you will see info like:

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
```

(continues on next page)

(continued from previous page)

```
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

You will then be asked whether you want the account data stored as a json file. Enter `y` to confirm and the file will be stored in `/Users/{username}/.local/share/aelf/keys/`.

Please make sure you remember the account data or the json file's location.

You will be required to set a password (referred to as * here):

```
Enter a password: ********
Confirm password: ********
```

For the sake of convenience, you are encouraged to keep this Terminal on the account info interface and open another Terminal to continue the following.

Run a Single Node

A single node runs aelf blockchain on one node. It is usually used to test the execution of contracts only.

1. Execute this command to start a Redis instance (skip this step if redis-server is already started):

```
redis-server
```

2. Open another Terminal and execute this command to change to aelf's directory:

```
cd ~/Desktop/Code/AElf
```

3. Execute this command to change to the `AElf.Launcher` directory:

```
cd src/AElf.Launcher
```

4. Modify the `appsettings.json` file: for novices, you can go to desktop -> Code -> AElf -> src -> AElf.Launcher and open the `appsettings.json` file in the editor to modify it (or, if you are familiar with Linux commands, you can run the `vim appsettings.json` command and modify the file in the command-line interface).

Find the account data you just created using `aelf-command create`.

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

Fill in the `NodeAccount` and `NodeAccountPassword` under `Account` using the `Address` and `password` you set in `appsettings.json`:

```
"Account": {
   "NodeAccount": "",
   "NodeAccountPassword": ""
 }
```

It may look like this when you complete it:

```
"Account": {
   "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
   "NodeAccountPassword": "********"
 },
```

Fill in the `InitialMineList` under `Consensus` using Public Key:

```
"Consensus": {
    "InitialMinerList": [],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

It may look like this when you complete it (make sure the key is bracketed):

```
"Consensus": {
    "InitialMinerList": [
↪"04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
↪"],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

If the IP and port for Redis have been changed, you can modify them under `ConnectionStrings` in `appsettings.json` (skip this step if they are not changed):

```
"ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
}
```

5. Execute `dotnet run`:

```
sudo dotnet run
```

The following output suggests successful execution:

```
2022-11-29 16:07:44,554 [.NET ThreadPool Worker] INFO  AElf.Kernel.
↪SmartContractExecution.Application.BlockExecutionResultProcessingService - Attach␣
↪blocks to best chain, best chain hash:
↪"f396756945d9bb883f81827ab36fcb0533d3c66f7062269700e49b74895*****", height: 177
```

If you want to check the node's block height and other block info, you can visit this page where you can access the API docs and interact with this single node.

To shut the node down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

Linux and Codespaces

Follow this doc to run an aelf single node in Linux and Codespaces and this will take around 20 minutes to complete.

Install aelf-command

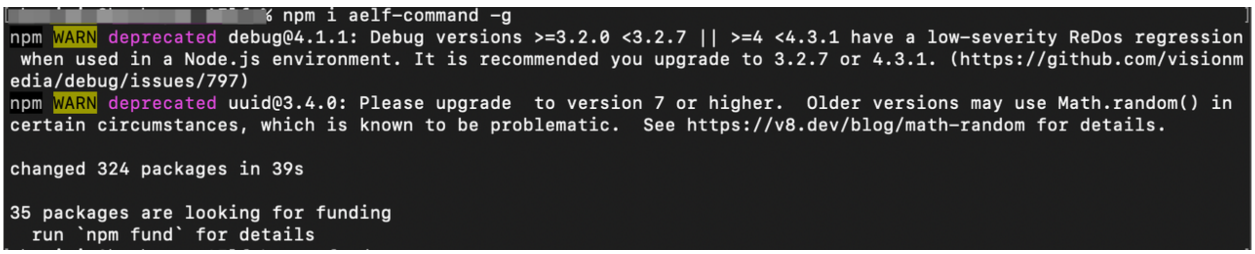
Execute this command to install aelf-command:

```
npm i aelf-command -g
```

The following output suggests successful installation:

```
+ aelf-command@0.1.44
added 314 packages from 208 contributors in 25.958s
```

You might see warnings like this due to differences in system configuration. Please ignore it:



Clone and Build aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command to create a directory:

```
mkdir ~/Desktop/Code
```

2. Execute this command to change the directory:

```
cd ~/Desktop/Code
```

3. Execute this command to clone aelf's code:

```
git clone https://github.com/AElfProject/AElf.git
```

4. Execute this command to change to aelf's directory:

```
cd AElf
```

5. Execute this command to restore aelf's files:

```
dotnet restore AElf.All.sln
```

6. Execute this command to build aelf's code (this will take several minutes):

```
dotnet build AElf.All.sln
```

The following output suggests successful building:

```
 xx Warning(s)
    0 Error(s)

Time Elapsed 00:15:59.77
```

Create an aelf Account

Execute this command:

```
aelf-command create
```

An aelf account will be automatically created and you will see info like:

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc0
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

You will then be asked whether you want the account data stored as a json file. Enter `y` to confirm and the file will be stored in `/root/.local/share/aelf/keys/`.

Please make sure you remember the account data or the json file's location.

You will be required to set a password (referred to as * here):

```
Enter a password: ********
Confirm password: ********
```

For the sake of convenience, you are encouraged to keep this Terminal on the account info interface and open another Terminal to continue the following.

Run a Single Node

A single node runs aelf blockchain on one node. It is usually used to test the execution of contracts only.

1. Execute this command to start a Redis instance (skip this step if redis-server is already started):

```
redis-server
```

2. Open another Terminal and execute this command to change to aelf's directory:

```
cd ~/Desktop/Code/AElf
```

3. Execute this command to change to the `AElf.Launcher` directory:

```
cd src/AElf.Launcher
```

4. Modify the `appsettings.json` file: for novices, you can go to desktop -> Code -> AElf -> src -> AElf.Launcher and open the `appsettings.json` file in the editor to modify it (or, if you are familiar with Linux commands, you can run the `vim appsettings.json` command and modify the file in the command-line interface).

Find the account data you just created using `aelf-command create`.

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc0
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

Fill in the `NodeAccount` and `NodeAccountPassword` under `Account` using the `Address` and `password` you set in `appsettings.json`:

```
"Account": {
   "NodeAccount": "",
   "NodeAccountPassword": ""
 }
```

It may look like this when you complete it:

```
"Account": {
   "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
   "NodeAccountPassword": "********"
 },
```

Fill in the `InitialMineList` under `Consensus` using Public Key:

```
"Consensus": {
    "InitialMinerList": [],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

It may look like this when you complete it (make sure the key is bracketed):

```
"Consensus": {
    "InitialMinerList": [
↪"04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
↪"],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

If the IP and port for Redis have been changed, you can modify them under `ConnectionStrings` in `appsettings.json` (skip this step if they are not changed):

```
"ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
}
```

Save the changes and keep them in the `AElf.Launcher` directory.

5. Execute `dotnet run`:

```
sudo dotnet run
```

The following output suggests successful execution:

```
2022-11-29 16:07:44,554 [.NET ThreadPool Worker] INFO  AElf.Kernel.
↪SmartContractExecution.Application.BlockExecutionResultProcessingService - Attach␣
↪blocks to best chain, best chain hash:
↪"f396756945d9bb883f81827ab36fcb0533d3c66f7062269700e49b74895*****", height: 177
```

If you want to check the node's block height and other block info, you can visit this page where you can access the API docs and interact with this single node.

To shut the node down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

Windows

Follow this doc to run an aelf single node on a Windows device and this will take around 20 minutes to complete.

Install aelf-command

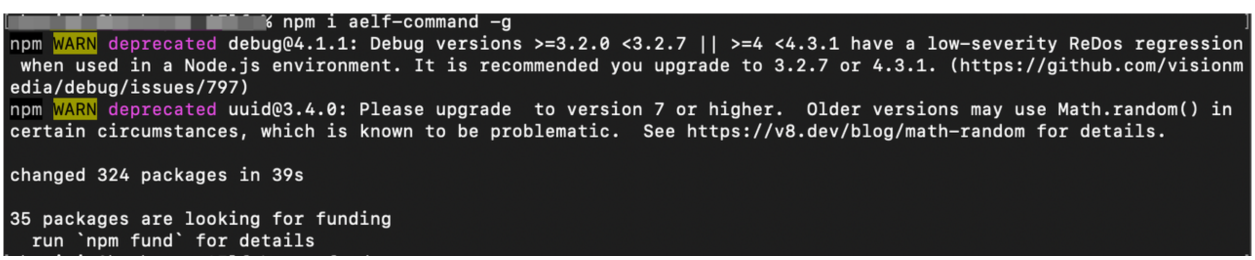
Execute npm command to install aelf-command:

```
npm i aelf-command -g
```

The following output suggests successful installation:

```
+ aelf-command@0.1.44
added 314 packages from 208 contributors in 25.958s
```

You might see warnings like this due to differences in system configuration. Please ignore it:



Clone and Build aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command in cmd or PowerShell to create a directory:

```
mkdir C:/Users/${username}/Desktop/Code
```

2. Execute this command to change the directory:

```
cd C:/Users/${username}/Desktop/Code
```

3. Execute this command to clone aelf's code:

```
git clone https://github.com/AElfProject/AElf.git
```

4. Execute this command to change to aelf's directory:

```
cd AElf
```

5. Execute this command to restore aelf's files:

```
dotnet restore AElf.All.sln
```

6. Execute this command to build aelf's code (this will take several minutes):

```
dotnet build AElf.All.sln
```

The following output suggests successful building:

```
 xx Warning(s)
    0 Error(s)

Time Elapsed 00:15:59.77
```

Create an aelf Account

Execute this command:

```
aelf-command create
```

An aelf account will be automatically created and you will see info like:

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc0
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

You will then be asked whether you want the account data stored as a json file. Enter `y` to confirm and the file will be stored locally.

Please make sure you remember the account data or the json file's location.

You will be required to set a password (referred to as * here):

```
Enter a password: ********
Confirm password: ********
```

For the sake of convenience, you are encouraged to keep this cmd or PowerShell on the account info interface and open another cmd or PowerShell to continue the following.

Run a Single Node

A single node runs aelf blockchain on one node. It is usually used to test the execution of contracts only.

1. Execute this command to start a Redis instance (skip this step if redis-server is already started):

```
redis-server
```

2. Open another cmd or PowerShell and execute this command to change to aelf's directory:

```
cd C:/Users/${username}/Desktop/Code
```

3. Execute this command to change to the `AElf.Launcher` directory:

```
cd src/AElf.Launcher
```

4. Modify the `appsettings.json` file: for novices, you can go to desktop -> Code -> AElf -> src -> AElf.Launcher and open the `appsettings.json` file in the editor to modify it (or you can run the `start appsettings.json` command and open the appsettings.json file in the editor).

Find the account data you just created using `aelf-command create`.

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : mirror among battle muffin cattle plunge tuition␣
↪buzz hip mad surround recall
AElf [Info]: Private Key         :␣
↪4bf625afea60e21aa5afcab5ea682b3dfb614941245698632d72a09ae13*****
AElf [Info]: Public Key          :␣
↪04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
AElf [Info]: Address             : 21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****
```

Fill in the `NodeAccount` and `NodeAccountPassword` under `Account` using the `Address` and `password` you set in `appsettings.json`:

```
"Account": {
   "NodeAccount": "",
   "NodeAccountPassword": ""
 }
```

It may look like this when you complete it:

```
"Account": {
   "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
   "NodeAccountPassword": "********"
 },
```

Fill in the `InitialMineList` under `Consensus` using Public Key:

```
"Consensus": {
    "InitialMinerList": [],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

It may look like this when you complete it (make sure the key is bracketed):

```
"Consensus": {
    "InitialMinerList": [
↪"04f9bb56a9eca921bd494e677307f0279c98f1d2ed6bdeaa6dd256878272eabd14e91ec61469d2a32ce5e63205930dabdc
↪"],
    "MiningInterval": 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 604800,
    "MinerIncreaseInterval": 31536000
  }
```

If the IP and port for Redis have been changed, you can modify them under `ConnectionStrings` in `appsettings.json` (skip this step if they are not changed):

```
"ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
}
```

Save the changes and keep them in the `AElf.Launcher` directory.

```
"ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
}
```

5. Execute `dotnet run`:

```
sudo dotnet run
```

The following output suggests successful execution:

```
2022-11-29 16:07:44,554 [.NET ThreadPool Worker] INFO  AElf.Kernel.
↪SmartContractExecution.Application.BlockExecutionResultProcessingService - Attach␣
↪blocks to best chain, best chain hash:
↪"f396756945d9bb883f81827ab36fcb0533d3c66f7062269700e49b74895*****", height: 177
```

If you want to check the node's block height and other block info, you can visit this page where you can access the API docs and interact with this single node.

To shut the node down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

2.2.2 Multi Nodes

macOS

Follow this doc to run aelf multi-nodes on a macOS device and this will take around 20 minutes to complete.

Run Multi-Nodes

This tutorial will guide you through how to run three nodes.

Publish aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command to create a directory:

```
mkdir ~/Desktop/Out
```

2. Execute this command to change the directory:

```
cd ~/Desktop/Code/AElf
```

3. Execute this command to publish aelf's code (this will take several minutes):

```
sudo dotnet publish AElf.All.sln /p:NoBuild=false --configuration Debug -o ~/Desktop/
↪Out
```

Configure Three Nodes

1. Execute this command three times to create three accounts: A, B, and C.

```
aelf-command create
```

Please make sure you remember their Public Keys and Addresses.

Create a directory for node configuration. This tutorial uses a directory on the desktop for reference.

2. Execute this command to create a directory:

```
mkdir ~/Desktop/Config
```

3. Execute this command to change the directory:

```
cd ~/Desktop/Config
```

4. Execute this command to create three new directories: `bp1`, `bp2`, and `bp3` in the "Config" directory and create their respective "keys" directories.

```
mkdir -p ~/Desktop/Config/bp1/keys

mkdir -p ~/Desktop/Config/bp2/keys

mkdir -p ~/Desktop/Config/bp3/keys
```

5. Copy account A, B, and C from `/Users/{username}/.local/share/aelf/keys/` to `bp1/keys`, `bp2/keys`, and `bp3/keys` respectively (If you can't find `.local`, you can use cmd + shift + g in Finder to designate the directories).

6. Execute this command to create `appsettings.json` files and `appsettings.MainChain.MainNet.json` files in directories `bp1`, `bp2`, and `bp3`:

```
cd ~/Desktop/Config/bp1;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json

cd ~/Desktop/Config/bp2;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json
```

(continues on next page)

(continued from previous page)

```
cd ~/Desktop/Config/bp3;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json
```

For appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug"
    }
  },
  "AllowedHosts": "*",
  "CorsOrigins": "*",
  "ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
  },
  "ChainId": "AELF",
  "IsMainChain" : true,
  "NetType": "MainNet",
  "Account": {
    "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
    "NodeAccountPassword": "********"
  },
  "Network": {
    "BootNodes": [],
    "ListeningPort": 7001,
    "NetAllowed": "",
    "NetWhitelist": []
  },
  "Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "http://*:8001/"
      }
    }
  },
  "Runner": {
    "BlackList": [],
    "WhiteList": []
  },
  "DeployServiceUrl": "",
  "Consensus": {
    "InitialMinerList" : [

↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed34acf875
↪",

↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd88048698
↪",

↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d545cc5c038
↪"
    ],
    "MiningInterval" : 4000,
    "StartTimestamp": 0,
```

(continues on next page)

(continued from previous page)

```
    "PeriodSeconds": 120
  },
  "BackgroundJobWorker":{
    "JobPollPeriod": 1
  }
}
```

For `appsettings.MainChain.MainNet.json`:

```
{
    "ChainId": "AELF",
    "TokenInitial": {
        "Symbol": "ELF",
        "Name": "elf token",
        "TotalSupply": 1000000000,
        "Decimals": 2,
        "IsBurnable": true,
        "DividendPoolRatio": 0.2
    },
    "ElectionInitial": {
        "LockForElection": 100000,
        "TimeEachTerm": 2,
        "BaseTimeUnit": 2,
        "MinimumLockTime": 1,
        "MaximumLockTime": 2000
    }
}
```

7. Modify the `appsettings.json` files in directory `bp1`, `bp2`, and `bp3` as instructed:

   1. Change the numbers following `db=` in `BlockchainDb` and `StateDb` under `ConnectionStrings`:

      1. `bp1`: redis://localhost:6379?db=1
      2. `bp2`: redis://localhost:6379?db=2
      3. `bp3`: redis://localhost:6379?db=3

   2. Replace `NodeAccount` and `NodeAccountPassword` under `Account` with `Address` and `password` in account A, B, and C.

   3. Fill in all three `InitialMineList` under `Consensus` using account A, B, and C's `Public Key`, keys separated with `,`:

      ```
      "Consensus": {
          "InitialMinerList" : [

       ↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed
       ↪",

       ↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd
       ↪",

       ↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d54
       ↪"
          ],
      ```

   4. In `bp1`, `BootNodes` is blank and `ListeningPort` is 7001. In `bp2`, `BootNodes` is `127.0.0.1:7001` (make sure to bracket it), and `ListeningPort` is 7002. In `bp3`, `BootNodes` are `127.`

0.0.1:7001 and 127.0.0.1:7002 (make sure to bracket them and separate them with ,) and ListeningPort is 7003.

5. Change the port numbers in Kestrel-EndPoints-Http-Url to 8001, 8002, and 8003 respectively (to ensure there is no conflict of ports).

8. Execute this command to start a Redis instance:

```
redis-server
```

Run Three Nodes

In this tutorial, code is published in ~/Desktop/Out and the three nodes are configured in ~/Desktop/Config.

Use redis-server to start a Redis instance.

We recommend you open three new Terminals to monitor the nodes' operation.

Execute this command to launch node 1:

```
cd ~/Desktop/Config/bp1;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 2:

```
cd ~/Desktop/Config/bp2;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 3:

```
cd ~/Desktop/Config/bp3;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

The three nodes run successfully if all Terminals show the following output:

```
2022-11-30 20:51:04,163 [.NET ThreadPool Worker] INFO  AElf.Kernel.Miner.Application.
↪MiningService - Generated block: { id:
↪"12f519e1601dd9f755a186b1370fd12696a8c080ea04465dadc*********2463", height: 25 },
↪previous: 5308de83c3585dbb4a097a9187a3b2f9b8584db4889d428484ca3e4df09e2860,
↪executed transactions: 2, not executed transactions 0
```

To shut the nodes down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

Linux and Codespaces

Follow this doc to run aelf multi-nodes in Linux and Codespaces and this will take around 20 minutes to complete.

Run Multi-Nodes

This tutorial will guide you through how to run three nodes.

Publish aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command to create a directory:

```
mkdir ~/Desktop/Code
```

2. Execute this command to change the directory:

```
cd ~/Desktop/Code/AElf
```

3. Execute this command to publish aelf's code (this will take several minutes):

```
sudo dotnet publish AElf.All.sln /p:NoBuild=false --configuration Debug -o ~/Desktop/
↪Out
```

Configure Three Nodes

1. Execute this command three times to create three accounts: A, B, and C.

```
aelf-command create
```

Please make sure you remember their Public Keys and Addresses.

Create a directory for node configuration. This tutorial uses a directory on the desktop for reference.

2. Execute this command to create a directory:

```
mkdir ~/Desktop/Config
```

3. Execute this command to change the directory:

```
cd ~/Desktop/Config
```

4. Execute this command to create three new directories: `bp1`, `bp2`, and `bp3` in the "Config" directory and create their respective "keys" directories.

```
mkdir -p ~/Desktop/Config/bp1/keys

mkdir -p ~/Desktop/Config/bp2/keys

mkdir -p ~/Desktop/Config/bp3/keys
```

5. Copy account A, B, and C from `/root/.local/share/aelf/keys/` to `bp1/keys`, `bp2/keys`, and `bp3/keys` respectively (If you can't find `.local`, you can use cmd + shift + g in Finder to designate the directories).

6. Execute this command to create `appsettings.json` files and `appsettings.MainChain.MainNet.json` files in directories `bp1`, `bp2`, and `bp3`:

```
cd ~/Desktop/Config/bp1;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json

cd ~/Desktop/Config/bp2;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json
```

(continues on next page)

(continued from previous page)

```
cd ~/Desktop/Config/bp3;touch appsettings.json;touch appsettings.MainChain.MainNet.
↪json
```

Copy the following templates to each file:

For `appsettings.json`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug"
    }
  },
  "AllowedHosts": "*",
  "CorsOrigins": "*",
  "ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
  },
  "ChainId": "AELF",
  "IsMainChain" : true,
  "NetType": "MainNet",
  "Account": {
    "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
    "NodeAccountPassword": "********"
  },
  "Network": {
    "BootNodes": [],
    "ListeningPort": 7001,
    "NetAllowed": "",
    "NetWhitelist": []
  },
  "Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "http://*:8001/"
      }
    }
  },
  "Runner": {
    "BlackList": [],
    "WhiteList": []
  },
  "DeployServiceUrl": "",
  "Consensus": {
    "InitialMinerList" : [

↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed34acf8752
↪",

↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd880486986
↪",

↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d545cc5c0382
↪"
    ],
```

(continues on next page)

(continued from previous page)

```
    "MiningInterval" : 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 120
  },
  "BackgroundJobWorker":{
    "JobPollPeriod": 1
  }
}
```

For `appsettings.MainChain.MainNet.json`:

```
{
    "ChainId": "AELF",
    "TokenInitial": {
        "Symbol": "ELF",
        "Name": "elf token",
        "TotalSupply": 1000000000,
        "Decimals": 2,
        "IsBurnable": true,
        "DividendPoolRatio": 0.2
    },
    "ElectionInitial": {
        "LockForElection": 100000,
        "TimeEachTerm": 2,
        "BaseTimeUnit": 2,
        "MinimumLockTime": 1,
        "MaximumLockTime": 2000
    }
}
```

7. Modify the `appsettings.json` files in directory `bp1`, `bp2`, and `bp3` as instructed:

   1. Change the numbers following `db=` in `BlockchainDb` and `StateDb` under `ConnectionStrings`:

      1. `bp1`: redis://localhost:6379?db=1
      2. `bp2`: redis://localhost:6379?db=2
      3. `bp3`: redis://localhost:6379?db=3

   2. Replace `NodeAccount` and `NodeAccountPassword` under `Account` with `Address` and `password` in account A, B, and C.

   3. Fill in all three `InitialMineList` under `Consensus` using account A, B, and C's `Public Key`, keys separated with`,`:

```
"Consensus": {
     "InitialMinerList" : [

↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed
↪",

↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd
↪",

↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d54
↪"
 ],
```

4. In bp1, BootNodes is blank and ListeningPort is 7001. In bp2, BootNodes is 127.0.0.1:7001 (make sure to bracket it), and ListeningPort is 7002. In bp3, BootNodes are 127.0.0.1:7001 and 127.0.0.1:7002 (make sure to bracket them and separate them with ,) and ListeningPort is 7003.
5. Change the port numbers in Kestrel-EndPoints-Http-Url to 8001, 8002, and 8003 respectively (to ensure there is no conflict of ports).

8. Execute this command to start a Redis instance:

```
redis-server
```

Run Three Nodes

In this tutorial, code is published in ~/Desktop/Out and the three nodes are configured in ~/Desktop/Config.

Use redis-server to start a Redis instance.

We recommend you open three new Terminals to monitor the nodes' operation.

Execute this command to launch node 1:

```
cd ~/Desktop/Config/bp1;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 2:

```
cd ~/Desktop/Config/bp2;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 3:

```
cd ~/Desktop/Config/bp3;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

The three nodes run successfully if all Terminals show the following output:

```
2022-11-30 20:51:04,163 [.NET ThreadPool Worker] INFO  AElf.Kernel.Miner.Application.
↪MiningService - Generated block: { id:
↪"12f519e1601dd9f755a186b1370fd12696a8c080ea04465dadc*********2463", height: 25 },␣
↪previous: 5308de83c3585dbb4a097a9187a3b2f9b8584db4889d428484ca3e4df09e2860,␣
↪executed transactions: 2, not executed transactions 0
```

To shut the nodes down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

Windows

Follow this doc to run aelf multi-nodes on a Windows device and this will take around 20 minutes to complete.

Run Multi-Nodes

This tutorial will guide you through how to run three nodes.

Publish aelf's Code

Create a directory. This tutorial uses a directory on the desktop for reference.

1. Execute this command to create a directory:

```
mkdir C:/Users/${username}/Desktop/Out
```

2. Execute this command to change the directory:

```
cd C:/Users/${username}/Desktop/Code/AElf
```

3. Execute this command to publish aelf's code (this will take several minutes):

Note: Be sure to replace `${username}` here with your user name.

```
sudo dotnet publish AElf.All.sln /p:NoBuild=false --configuration Debug -o C:/Users/$
↪{username}/Desktop/Out
```

Configure Three Nodes

1. Execute this command three times to create three accounts: A, B, and C.

```
aelf-command create
```

Please make sure you remember their Public Keys and Addresses.

Create a directory for node configuration. This tutorial uses a directory on the desktop for reference.

2. Execute this command to create a directory:

```
mkdir C:/Users/${username}/Desktop/Config
```

3. Execute this command to change the directory:

```
cd C:/Users/${username}/Desktop/Config
```

4. Execute this command to create three new directories: `bp1`, `bp2`, and `bp3` in the "Config" directory and create their respective "keys" directories.

```
mkdir -p C:/Users/${username}/Desktop/Config/bp1/keys

mkdir -p C:/Users/${username}/Desktop/Config/bp2/keys

mkdir -p C:/Users/${username}/Desktop/Config/bp3/keys
```

5. Copy account A, B, and C from their json files to `bp1/keys`, `bp2/keys`, and `bp3/keys` respectively.

6. Execute this command to create `appsettings.json` files and `appsettings.MainChain.MainNet.json` files in directories `bp1`, `bp2`, and `bp3`:

```
cd C:/Users/${username}/Desktop/Config/bp1;touch appsettings.json;touch appsettings.
↪MainChain.MainNet.json

cd C:/Users/${username}/Desktop/Config/bp2;touch appsettings.json;touch appsettings.
↪MainChain.MainNet.json
```

(continues on next page)

(continued from previous page)

```
cd C:/Users/${username}/Desktop/Config/bp3;touch appsettings.json;touch appsettings.
↪MainChain.MainNet.json
```

Copy the following templates to each file:

For `appsettings.json`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug"
    }
  },
  "AllowedHosts": "*",
  "CorsOrigins": "*",
  "ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
  },
  "ChainId": "AELF",
  "IsMainChain" : true,
  "NetType": "MainNet",
  "Account": {
    "NodeAccount": "21qciGwcaowwBttKMjMk86AW6WajhcodSHytY1vCyZb7p*****",
    "NodeAccountPassword": "********"
  },
  "Network": {
    "BootNodes": [],
    "ListeningPort": 7001,
    "NetAllowed": "",
    "NetWhitelist": []
  },
  "Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "http://*:8001/"
      }
    }
  },
  "Runner": {
    "BlackList": [],
    "WhiteList": []
  },
  "DeployServiceUrl": "",
  "Consensus": {
    "InitialMinerList" : [

↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed34acf8752
↪",

↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd88048698
↪",

↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d545cc5c0382
↪"
    ],
```

(continues on next page)

(continued from previous page)

```
    "MiningInterval" : 4000,
    "StartTimestamp": 0,
    "PeriodSeconds": 120
  },
  "BackgroundJobWorker":{
    "JobPollPeriod": 1
  }
}
```

For `appsettings.MainChain.MainNet.json`:

```
{
    "ChainId": "AELF",
    "TokenInitial": {
        "Symbol": "ELF",
        "Name": "elf token",
        "TotalSupply": 1000000000,
        "Decimals": 2,
        "IsBurnable": true,
        "DividendPoolRatio": 0.2
    },
    "ElectionInitial": {
        "LockForElection": 100000,
        "TimeEachTerm": 2,
        "BaseTimeUnit": 2,
        "MinimumLockTime": 1,
        "MaximumLockTime": 2000
    }
}
```

7. Modify the `appsettings.json` files in directory `bp1`, `bp2`, and `bp3` as instructed:
    1. Change the numbers following `db=` in `BlockchainDb` and `StateDb` under `ConnectionStrings`:
        1. `bp1`: redis://localhost:6379?db=1
        2. `bp2`: redis://localhost:6379?db=2
        3. `bp3`: redis://localhost:6379?db=3
    2. Replace `NodeAccount` and `NodeAccountPassword` under `Account` with `Address` and `password` in account A, B, and C.
    3. Fill in all three `InitialMineList` under `Consensus` using account A, B, and C's `Public Key`, keys separated with `,`:

```
"Consensus": {
    "InitialMinerList" : [

↪"04884d9563b3b67a589e2b9b47794fcfb3e15fa494053088dd0dc8a909dd72bfd24c43b0e2303d631683acaed
↪",

↪"045670526219d73154847b1e9367be9af293601793c9f7e34a96336650c9c1104a4aac9aaee960af00e775dcd
↪",

↪"046a5913eae5fee3da9ee33604119f025a0ad45575dfed1257eff5da2c24e629845b1e1a131c5da8751971d54
↪"
    ],
```

4. In `bp1`, `BootNodes` is blank and `ListeningPort` is 7001. In `bp2`, `BootNodes` is `127.0.0.1:7001` (make sure to bracket it), and `ListeningPort` is 7002. In `bp3`, `BootNodes` are `127.0.0.1:7001` and `127.0.0.1:7002` (make sure to bracket them and separate them with `,`) and `ListeningPort` is 7003.
5. Change the port numbers in `Kestrel-EndPoints-Http-Url` to 8001, 8002, and 8003 respectively (to ensure there is no conflict of ports).

8. Execute this command to start a Redis instance:

```
redis-server
```

Run Three Nodes

In this tutorial, code is published in `C:/Users/${username}/Desktop/Out` and the three nodes are configured in `C:/Users/${username}/Desktop/Config`.

Use `redis-server` to start a Redis instance.

We recommend you open three new terminals to monitor the nodes' operation.

Execute this command to launch node 1:

```
cd ~/Desktop/Config/bp1;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 2:

```
cd ~/Desktop/Config/bp2;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

Execute this command to launch node 3:

```
cd ~/Desktop/Config/bp3;dotnet ~/Desktop/Out/AElf.Launcher.dll
```

The three nodes run successfully if all Terminals show the following output:

```
2022-11-30 20:51:04,163 [.NET ThreadPool Worker] INFO  AElf.Kernel.Miner.Application.
↪MiningService - Generated block: { id:
↪"12f519e1601dd9f755a186b1370fd12696a8c080ea04465dadc*********2463", height: 25 },␣
↪previous: 5308de83c3585dbb4a097a9187a3b2f9b8584db4889d428484ca3e4df09e2860,␣
↪executed transactions: 2, not executed transactions 0
```

To shut the nodes down, please use control + c on your keyboard.

If you don't want to save the data, you can execute this command to delete all:

```
redis-cli flushall
```

CHAPTER 3

Smart Contract Development

3.1 Greeter Contract

3.1.1 Smart contract implementation

This article will guide you through how to use **AElf Boilerplate** to implement a smart contract. It takes an example on the **Greeter** contract that's already included in Boilerplate. Based on the concepts this article presents, you'll be able to create your own basic contract.

Greeter contract

The following content will walk you through the basics of writing a smart contract; this process contains essentially four steps:

- **create the project**: generate the contract template using **AElf Boilerplate**'s code generator.
- **define the contract and its types**: the methods and types needed in your contract should be defined in a protobuf file, following typical protobuf syntax.
- **generate the code**: build the project to generate the base contract code from the proto definition.
- **extend the generated code**: implement the logic of the contract methods.

The `Greeter` contract is a very simple contract that exposes a `Greet` method that simply logs to the console and returns a "Hello World" message and a more sophisticated `GreetTo` method that records every greeting it receives and returns the greeting message as well as the time of the greeting.

This tutorial shows you how to develop a smart contract with the C# contract SDK; you can find you more *here*. Boilerplate will automatically add the reference to the SDK.

Create the project

With **AElf Boilerplate**'s code generator, you can easily and quickly set up a contract project. See here for details.

Defining the contract

After creating the contract project, you can define the methods and types of your contract. **AElf** defines smart contracts as services that are implemented using gRPC and Protobuf. The definition contains no logic; at build time the proto file is used to generate C# classes that will be used to implement the logic and state of the contract.

We recommend putting the contract's definition in Boilerplate's **protobuf** folder so that it can easily be included in the build/generation process and also that you name the contract with the following syntax **contract_name_contract.proto**:

```
.
└── Boilerplate
    └── chain
        └── protobuf
            ├── aelf
            │   ├── options.proto // contract options
            │   └── core.proto    // core blockchain types
            ├── greeter_contract.proto
            ├── another_contract.proto
            ├── token_contract.proto // system contracts
            ├── acs0.proto // AElf contract standard
            └── ...
```

The "protobuf" folder already contains a certain amount of contract definitions, including tutorial examples, system contracts. You'll also notice it contains AElf Contract Standard definitions that are also defined the same way as contracts. Lastly, it also contains **options.proto** and **core.proto** that contain fundamental types for developing smart contracts, more on this later.

Best practices:

- place your contract definition in Boilerplate's **protobuf** folder.
- name your contract with **contractname_contract.proto**, all lower case.

Now let's take a look a the Greeter contract's definition:

```
// protobuf/greeter_contract.proto

syntax = "proto3";

import "aelf/options.proto";

import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
import "google/protobuf/wrappers.proto";

option csharp_namespace = "AElf.Contracts.Greeter";

service GreeterContract {
    option (aelf.csharp_state) = "AElf.Contracts.Greeter.GreeterContractState";

    // Actions
    rpc Greet (google.protobuf.Empty) returns (google.protobuf.StringValue) { }
    rpc GreetTo (google.protobuf.StringValue) returns (GreetToOutput) { }

    // Views
    rpc GetGreetedList (google.protobuf.Empty) returns (GreetedList) {
        option (aelf.is_view) = true;
    }
```

(continues on next page)

(continued from previous page)

```
}

message GreetToOutput {
    string name = 1;
    google.protobuf.Timestamp greet_time = 2;
}

message GreetedList {
    repeated string value = 1;
}
```

Above is the full definition of the contract; it is mainly composed of three parts:

- **imports**: the dependencies of your contract.
- **the service definition**: the methods of your contract.
- **types**: some custom defined types used by the contract.

Let's have a deeper look at the three different parts.

Syntax, imports and namespace

```
syntax = "proto3";

import "aelf/options.proto";

import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
import "google/protobuf/wrappers.proto";

option csharp_namespace = "AElf.Contracts.Greeter";
```

The first line specifies the syntax that this protobuf file uses, we recommend you always use **proto3** for your contracts. Next, you'll notice that this contract specifies some imports, let's analyze them briefly:

- **aelf/options.proto** : contracts can use AElf specific options; this file contains the definitions. One example is the **is_view** options that we will use later.
- **empty.proto, timestamp.proto and wrappers.proto** : these are proto files imported directly from protobuf's library. They are useful for defining things like an empty return value, time, and wrappers around some common types such as string.

The last line specifies an option that determines the target namespace of the generated code. Here the generated code will be in the `AElf.Contracts.Greeter` namespace.

The service definition

```
service GreeterContract {
    option (aelf.csharp_state) = "AElf.Contracts.Greeter.GreeterContractState";

    // Actions
    rpc Greet (google.protobuf.Empty) returns (google.protobuf.StringValue) { }
    rpc GreetTo (google.protobuf.StringValue) returns (GreetToOutput) { }
```

(continues on next page)

(continued from previous page)

```
    // Views
    rpc GetGreetedList (google.protobuf.Empty) returns (GreetedList) {
        option (aelf.is_view) = true;
    }
}
```

The first line here uses the `aelf.csharp_state` option to specify the name (full name) of the state class. This means that the state of the contract should be defined in the `GreeterContractState` class under the `AElf.Contracts.Greeter` namespace.

Next, two **action** methods are defined: `Greet` and `GreetTo`. A contract method is defined by three things: the **method name**, the **input argument(s) type(s)** and the **output type**. For example, `Greet` requires that the input type is `google.protobuf.Empty` that is used to specify that this method takes no arguments and the output type will be a google.protobuf.StringValue is a traditional string. As you can see with the `GreetTo` method, you can use custom types as input and output of contract methods.

The service also defines a **view** method, that is, a method used only to query the contracts state, and that has no side effect on the state. For example, the definition of `GetGreetedList` uses the **aelf.is_view** option to make it a view method.

Best practice:

- use **google.protobuf.Empty** to specify that a method takes no arguments (import `google/protobuf/empty.proto`).
- use **google.protobuf.StringValue** to use a string (import `google/protobuf/wrappers.proto`).
- use the **aelf.is_view** option to create a view method (import `aelf/options.proto`).
- use the **aelf.csharp_state** to specify the namespace of your contracts state (import `aelf/options.proto`).

Custom types

```
message GreetToOutput {
    string name = 1;
    google.protobuf.Timestamp greet_time = 2;
}

message GreetedList {
    repeated string value = 1;
}
```

The protobuf file also includes the definition of two custom types. The **GreetToOutput** is the type returned by the `GreetTo` method and `GreetedList` is the return type of the `GetGreetedList` view method. You'll notice the **repeated** keyword the `GreetedList` message. This is protobuf syntax to represent a collection.

Best practice:

- use **google.protobuf.Timestamp** to represent a point in time (import `google/protobuf/timestamp.proto`).
- use **repeated** to represent a collection of items of the same type.

Extend the generated code

After defining and generating the code from the definition, the contract author extends the generated code to implement the logic of his contract. Two files are presented here:

- **GreeterContract**: the actual implementation of the logic, it inherits from the contract base generated by protobuf.
- **GreeterContractState**: the state class that contains properties for reading and writing the state. This class inherits the `ContractState` class from the C# SDK.

```
// contract/AElf.Contracts.GreeterContract/GreeterContract.cs

using Google.Protobuf.WellKnownTypes;

namespace AElf.Contracts.Greeter
{
    public class GreeterContract : GreeterContractContainer.GreeterContractBase
    {
        public override StringValue Greet(Empty input)
        {
            Context.LogDebug(() => "Hello World!");
            return new StringValue {Value = "Hello World!"};
        }

        public override GreetToOutput GreetTo(StringValue input)
        {
            // Should not greet to empty string or white space.
            Assert(!string.IsNullOrWhiteSpace(input.Value), "Invalid name.");

            // State.GreetedList.Value is null if not initialized.
            var greetList = State.GreetedList.Value ?? new GreetedList();

            // Add input.Value to State.GreetedList.Value if it's new to this list.
            if (!greetList.Value.Contains(input.Value))
            {
                greetList.Value.Add(input.Value);
            }

            // Update State.GreetedList.Value by setting it's value directly.
            State.GreetedList.Value = greetList;

            Context.LogDebug(() => "Hello {0}!", input.Value);

            return new GreetToOutput
            {
                GreetTime = Context.CurrentBlockTime,
                Name = input.Value.Trim()
            };
        }

        public override GreetedList GetGreetedList(Empty input)
        {
            return State.GreetedList.Value ?? new GreetedList();
        }
    }
}
```

```
// contract/AElf.Contracts.GreeterContract/GreeterContractState.cs

using AElf.Sdk.CSharp.State;

 namespace AElf.Contracts.Greeter
```

(continues on next page)

(continued from previous page)

```
{
    public class GreeterContractState : ContractState
    {
        public SingletonState<GreetedList> GreetedList { get; set; }
    }
}
```

Let's briefly explain what is happening in the `GreetTo` method:

Asserting

```
Assert(!string.IsNullOrWhiteSpace(input.Value), "Invalid name.");
```

When writing a smart contract, it is often useful (and recommended) to validate the input. AElf smart contracts can use the `Assert` method defined in the base smart contract class to implement this pattern. For example, here, the method validates that the input string is null or composed only of white spaces. If the condition is false, this line will abort the execution of the transaction.

Accessing and saving state

```
var greetList = State.GreetedList.Value ?? new GreetedList();
...
State.GreetedList.Value = greetList;
```

From within the contract methods, you can easily access the contracts state through the `State` property of the contract. Here the state property refers to the `GreeterContractState` class in which is defined the `GreetedList` collection. The second effectively updates the state (this is needed; otherwise, the method would have no effect on the state).

Note that because the `GreetedList` type is wrapped in a `SingletonState` you have to use the `Value` property to access the data (more on this later).

Logging

```
Context.LogDebug(() => "Hello {0}!", input.Value);
```

It is also possible to log from smart contract methods. The above example will log "Hello" and the value of the input. It also prints useful information like the ID of the transaction. It will print in the console log if you launch the node with DEBUG mode. This is only for debug use and has no impacts on state at all.

More on state

As a reminder, here is the state definition in the contract (we specified the name of the class and a type) as well as the custom type `GreetedList`:

```
service GreeterContract {
    option (aelf.csharp_state) = "AElf.Contracts.Greeter.GreeterContractState";
    ...
}
```

(continues on next page)

(continued from previous page)

```
// ...

message GreetedList {
    repeated string value = 1;
}
```

The `aelf.csharp_state` option allows the contract author to specify in which namespace and class name the state will be. To implement a state class, you need to inherit from the `ContractState` class that is contained in the C# SDK (notice the `using` statement here below).

Below is the state class that we saw previously:

```
using AElf.Sdk.CSharp.State;

 namespace AElf.Contracts.Greeter
 {
    public class GreeterContractState : ContractState
    {
        public SingletonState<GreetedList> GreetedList { get; set; }
    }
 }
```

The state uses the custom `GreetedList` type, which was generated from the Protobuf definition at build time and contained exactly one property: a singleton state of type `GreetedList`.

The `SingletonState` is part of the C# SDK and is used to represent exactly **one** value. The value can be of any type, including collection types. Here we only wanted our contract to store one list (here a list of strings).

Note that you have to wrap your state types in a type like `SingletonState` (others are also available like `MappedState`) because behind the scene, they implement the state read and write operations.

3.1.2 Unit testing a contract

The previous article exposed how to add the proto definition and implement the logic of your contract. This article expands on the previous and will show you how to test your contract.

AElf Contract TestKit is a testing framework specifically used to test AElf smart contracts. With this framework, you can simulate the execution of a transaction by constructing a stub of a smart contract and using the methods provided by the Stub instance (corresponding to the contract's Action methods) and query (corresponding to the View methods of the contract), and then get the transaction execution results in the test case.

Test project

AElf Boilerplate's code generator has automatically generated test project for you, you just need to add your test cases.

As you can see, tests are placed in the **test** folder. Each test folder usually contains a project file (.csproj) and at least four .cs files. The project file is a basic C# xUnit test project file, to which we've added some references.

```
.
└── chain
    ├── contract
    ├── protobuf
    ├── src
```

(continues on next page)

(continued from previous page)

```
    └── test
        ├── AElf.Contracts.GreeterContract.Tests
            ├── AElf.Contracts.GreeterContract.Tests.csproj // xUnit test project
            ├── GreeterContractTestBase.cs
            ├── GreeterContractTestModule.cs
            ├── GreeterContractTests.cs
            └── GreeterContractInitializationProvider.cs
        └── ...
```

Test your contract

Now for the easy part, the test class only needs to inherit from the test base. After this you can go ahead and create the test cases you need.

GreeterContractTest.cs

```
public class GreeterContractTests : GreeterContractTestBase
{
    // declare the method as a xUnit test method
    [Fact]
    public async Task GreetTest()
    {
        // Use the contracts stub to call the 'Greet' method and get a reference to
        // the transaction result.
        var txResult = await GetGreeterContractStub(_defaultKeyPair).Greet.
↪SendAsync(new Empty());

        // check that the transaction was mined
        txResult.TransactionResult.Status.ShouldBe(TransactionResultStatus.Mined);

        // parse the result (return from the contract)
        var text = new StringValue();
        text.MergeFrom(txResult.TransactionResult.ReturnValue);

        // check that the value is correct
        text.Value.ShouldBe("Hello World!");
    }

    // ...
}
```

From the previous code snippet you can note several things:

- the test case is a classic xUnit test class.
- you can use the contracts stub to call the contract and check returns.

Feel free to have a look at the full test class in the Boilerplate source code.

3.1.3 Run the node

Next you can run Boilerplate (and it's an internal node). This will automatically deploy the Greeter contract. Open a terminal in the root Boilerplate directory and navigate to the launcher project:

```
cd chain/src/AElf.Boilerplate.GreeterContract.Launcher
```

Next, run the node:

```
dotnet run AElf.Boilerplate.GreeterContract.Launcher.csproj
```

From here, you should see the build and eventually the nodes logs.

Boilerplate will deploy your contract when the node starts. You can call the Boilerplate node API:

```
aelf-command get-chain-status
? Enter the the URI of an AElf node: http://127.0.0.1:1235
✓ Succeed
{
  "ChainId": "AELF",
  "Branches": {
    "6032b553ec9a5c81713cf8410f426dfc1ca0f43e64d56f527fc7a9c60b90e694": 3073
  },
  "NotLinkedBlocks": {},
  "LongestChainHeight": 3073,
  "LongestChainHash":
↪"6032b553ec9a5c81713cf8410f426dfc1ca0f43e64d56f527fc7a9c60b90e694",
  "GenesisBlockHash":
↪"c3bddca1909ebf37b95be7f26b990e07916790913e0f48da1a831b3c777d59ff",
  "GenesisContractAddress": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
  "LastIrreversibleBlockHash":
↪"85fee024d156de3be665c296c567423026e0e3369ad7dc5ee81dbb2a15dfe2f2",
  "LastIrreversibleBlockHeight": 3042,
  "BestChainHash": "6032b553ec9a5c81713cf8410f426dfc1ca0f43e64d56f527fc7a9c60b90e694",
  "BestChainHeight": 3073
}
```

This enables further testing of the contract, including testing it from a dApp.

3.1.4 Front end

This tutorial will show you how to develop a front-end app (JavaScript in our case) that will demonstrate how to interact with a contract that was developed with Boilerplate.

At the top-level Boilerplate contains two folders:

- chain : used for developing the contracts.
- web : used for developing the front-end.

The **web** folder already contains some projects that can serve as examples. This tutorial presents a front-end for the Greeter contract shown in the previous tutorials.

Run the front-end

After you run Boilerplate, open another terminal at the repo's root and navigate to the **greeter** project:

```
cd web/greeter
```

From here, you can install and run the Greeter's front end:

```
npm i
npm start
```

And a page will be opened by webpack in your default browser.

Front-end code

The code is straightforward, it uses aelf-sdk + webpack. You can check out more **here**.

Warning: be careful, this code is in no way production-ready and is for demonstration purposes only.

It demonstrates the following capabilities of the js sdk:

- getting the chain status.
- getting a contract object.
- calling a contract method.
- calling a view method.

Getting the chain status

The following code snippet shows how to call the nodes API to get the chains status:

```
aelf.chain.getChainStatus()
    .then(res => {
        if (!res) {
            throw new Error('Error occurred when getting chain status');
        }
        // use the chain status
    })
    .catch(err => {
        console.log(err);
    });
```

For more information about the chain status API : *GET /api/blockChain/chainStatus*.

As we will see next, the chain status is very useful for retrieving the genesis contract.

getting a contract object

The following code snippet shows how to get a contract object with the js-sdk:

```
async function getContract(name, walletInstance) {

    // if not loaded, load the genesis
    if (!genesisContract) {
        const chainStatus = await aelf.chain.getChainStatus();
        if (!chainStatus) {
            throw new Error('Error occurred when getting chain status');
        }
        genesisContract = await aelf.chain.contractAt(chainStatus.
↪GenesisContractAddress, walletInstance);
    }

```

(continues on next page)

(continued from previous page)

```
    // if the contract is not already loaded, get it by name.
    if (!contract[name]) {
        const address = await genesisContract.GetContractAddressByName.
↪call(sha256(name));
        contract = {
            ...contract,
            [name]: await aelf.chain.contractAt(address, walletInstance)
        };
    }
    return contract[name];
}
```

As seen above, the following steps will enable you to build a contract object:

- use **getChainStatus** to get the genesis contract's address.
- use **contractAt** to build an instance of the genesis contract.
- use the genesis contract to get the address of the greeter contract with the **GetContractAddressByName** method.
- with the address use **contractAt** again to build a greeter contract object.

Once you have a reference to the greeter contract, you can use it to call the methods.

calling a contract method

The following snippet shows how to send a transaction to the contract:

```
    greetToButton.onclick = () => {

        getContract('AElf.ContractNames.Greeter', wallet)
            .then(greeterContract => greeterContract.GreetTo({
                value: "SomeName"
            }))
            .then(tx => pollMining(tx.TransactionId))
            .then(ret => {
                greetToResponse.innerHTML = ret.ReadableReturnValue;
            })
            .catch(err => {
                console.log(err);
            });
    };
```

Here the **getContract** retrieves the greeter contract instance. On the instance it calls **GreetTo** that will send a transaction to the node. The **pollMining** method is a helper method that will wait for the transaction to be mined. After mined the transaction results, **ReadableReturnValue** will be used to see the result.

calling a view method

The following snippet shows how to call a view method on the contract:

```
    getGreeted.onclick = () => {

        getContract('AElf.ContractNames.Greeter', wallet)
```

(continues on next page)

(continued from previous page)

```
            .then(greeterContract => greeterContract.GetGreetedList.call())
            .then(ret => {
                greeted.innerHTML = JSON.stringify(ret, null, 2);
            })
            .catch(err => {
                console.log(err);
            });
    };
```

Here the **getContract** retrieves the greeter contract instance. On the instance, it calls **GetGreetedList** with ".call" appended to it, which will indicate a read-only execution (no broadcasted transaction).

3.2 Smart contract deployment

After the contract has been compiled, the user must register this contract with the blockchain. Generally, to deploy a contract, there must be transactions sent to Smart contract zero, which is one of AElf's genesis contracts. The node will then broadcast these transactions, and it will eventually get included in a block when the block gets executed the smart contract will be deployed.

For contract deployment, what matters is the `ContractDeploymentAuthorityRequired` option in the `ContractOptions` for this network. It is determined since the launch of the chain.

- if `ContractDeploymentAuthorityRequired` is false, anyone can directly deploy contract with transaction
- Only account with specific authority is permitted to deploy contract if `ContractDeploymentAuthorityRequired` is true

This part will introduce contract deployment pipeline for different chain type on AElf mainnet/testnet/customnet network.

3.2.1 Authority check

`ContractDeploymentAuthorityRequired` is false

Anyone can directly deploy contract with transaction if `ContractDeploymentAuthorityRequired` is false. It is usually set as false especially when it is for contract unit test or custom network.

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

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

The return value of this transaction indicates the address of the deployed contract. Note that you should specific 0 as category for c# contract and provide your contract dll bytes.

`ContractDeploymentAuthorityRequired` is true

`ContractDeploymentAuthorityRequired` is always true when it comes to public networks(Mainnet/Testnet). And contract pipelines are distinguished for different chain types. But for sure, no one can directly deploy.

For public network, no matter it is mainnet or testnet, things are going more complex. No one can directly deploy on the chain but few authorities have the permission to propose.

- Main Chain: only current miners have the permission to propose contract
- Exclusive Side Chain: only side chain creator are allowed to propose contract
- Shared Side Chain: anyone can propose contract

And contract proposing steps are provided as below

```
rpc ProposeNewContract (ContractDeploymentInput) returns (aelf.Hash) {
}
message ContractDeploymentInput {
    sint32 category = 1;
    bytes code = 2;
}

message ContractProposed
{
    option (aelf.is_event) = true;
    aelf.Hash proposed_contract_input_hash = 1;
}
```

Event `ContractProposed` will be fired containing `proposed_contract_input_hash` and this will also trigger the first proposal for one parliament organization, which is specified as contract deployment controller since the beginning of the chain. This proposal would be expired in 24 hours. Once the proposal can be released (refer to *Parliament contract* for detail), proposer should send transaction to

```
rpc ReleaseApprovedContract (ReleaseContractInput) returns (google.protobuf.
↪Empty) {
}
message ReleaseContractInput {
    aelf.Hash proposal_id = 1;
    aelf.Hash proposed_contract_input_hash = 2;
}
```

This will trigger the second proposal for one parliament organization, which is specified as contract code-check controller since the beginning of the chain. This proposal would be expired in 10 min. Once the proposal can be released, proposer should send transaction to

```
rpc ReleaseCodeCheckedContract (ReleaseContractInput) returns (google.protobuf.
↪Empty) {
}
message ReleaseContractInput {
    aelf.Hash proposal_id = 1;
    aelf.Hash proposed_contract_input_hash = 2;
}

message ContractDeployed
{
    option (aelf.is_event) = true;
    aelf.Address author = 1 [(aelf.is_indexed) = true];
    aelf.Hash code_hash = 2 [(aelf.is_indexed) = true];
    aelf.Address address = 3;
    int32 version = 4;
    aelf.Hash Name = 5;
}
```

Finally, the contract would be deployed. Event `ContractDeployed` containing new contract address will be fired and it is available in `TransactionResult.Logs`.

3.2.2 Use aelf-command send or aelf-command proposal to deploy

If you set `ContractDeploymentAuthorityRequired: true` in appsetting.json, please use aelf-command proposal.

```
$ aelf-command send <GenesisContractAddress> DeploySmartContract # aelf-command send
$ aelf-command send <GenesisContractAddress> ProposeNewContract # aelf-command␣
↪proposal
# Follow the instructions
```

- You must input contract method parameters in the prompting way, note that you can input a relative or absolute path of contract file to pass a file to aelf-command, aelf-command will read the file content and encode it as a base64 string.
- After call ProposeNewContract, you need to wait for the organization members to approve your proposal and you can release your proposal by calling `ReleaseApprovedContract` and `ReleaseCodeCheckedContract` in this order.

The deploy command(This command has been deprecated)

The **deploy** command on the cli will help you deploy the contract:

```
aelf-command deploy <category> <code>
```

The deploy command will create and send the transaction to the nodes RPC. Here the **code** is the path to the compiled code. This will be embedded in the transaction as a parameter to the **DeploySmartContract** method on smart contract zero. The command will return the ID of the transaction that was sent by the command. You will see in the next section how to use it.

verify the result

When the deployment transaction gets included in a block, the contract should be deployed. To check this, you can use the transaction ID returned by the deploy command. When the status of the transaction becomes **mined**: `"Status": "Mined"`, then the contract is ready to be called.

The **ReadableReturnValue** field indicates the address of the deployed contract. You can use this address to call the contract methods.

CHAPTER 4

AElf Blockchain Boot Sequence

This section mainly explains how the AElf Blockchain starts from the initial nodes, and gradually replaces the initial nodes with true production nodes through elections, thus completing the complete process of AElf Blockchain startup.

4.1 Start initial nodes

We need to start at least one or more initial nodes to start the AElf Blockchain, and 1-5 initial nodes are recommended.

In the Getting Started section, we described the steps to start multiple nodes, you can follow the *Running multi-nodes with Docker* to complete the initial nodes startup (this section also takes the example of starting three initial nodes).

Since the default period of election time is 604800 seconds(7 days), if you want to see the result of the election more quickly, modify the configuration file appsettings.json before starting the boot nodes to set the PeriodSeconds to smaller:

```
{
  "Consensus": {
    "PeriodSeconds": 604800
  },
}
```

4.2 Run full node

4.2.1 Create an account for the full node:

```
aelf-command create

AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : major clap hurdle hammer push slogan ranch quantum␣
↪reunion hope enroll repeat
```

(continues on next page)

(continued from previous page)

```
AElf [Info]: Private Key         :␣
↪2229945cf294431183fd1d8101e27b17a1a590d3a1f7f2b9299850b24262ed8a
AElf [Info]: Public Key          :␣
↪04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9ae
AElf [Info]: Address             : Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC
```

4.2.2 Start full node:

The startup steps for the full node are similar to the initial node startup, but the configuration file section notes that the InitialMinerList needs to be consistent with the initial node:

```
{
  "InitialMinerList" : [

↪"0499d3bb14337961c4d338b9729f46b20de8a49ed38e260a5c19a18da569462b44b820e206df8e848185dac6c139f0539
↪",

↪"048397dfd9e1035fdd7260329d9492d88824f42917c156aef93fd7c2e3ab73b636f482b8ceb5cb435c556bfa067445a86e
↪",

↪"041cc962a51e7bbdd829a8855eca8a03fda708fdf31969251321cb31edadd564bf3c6e7ab31b4c1f49f0f206be81dbe68a
↪"
  ],
}
```

4.2.3 Full node started successfully:

By checking the current node state, it can be seen that the full node is synchronizing, and the BestChainHeight and the LastIrreversibleBlockHeight are growing up. After catching up with the height of the initial node, the subsequent steps can be carried out.

```
aelf-command get-chain-status

{
  "ChainId": "AELF",
  "Branches": {
    "fb749177c2f43db8c7d73ea050240b9f870c40584f044b13e7ec146c460b0eff": 2449
  },
  "NotLinkedBlocks": {},
  "LongestChainHeight": 2449,
  "LongestChainHash":
↪"fb749177c2f43db8c7d73ea050240b9f870c40584f044b13e7ec146c460b0eff",
  "GenesisBlockHash":
↪"ea9c0b026bd638ceb38323eb71174814c95333e39c62936a38c4e01a8f18062e",
  "GenesisContractAddress": "pykr77ft9UUKJZLVq15wCH8PinBSjVRQ12sD1Ayq92mKFsJ1i",
  "LastIrreversibleBlockHash":
↪"66638f538038bd56357f3cf205424e7393c5966830ef0d16a75d4a117847e0bc",
  "LastIrreversibleBlockHeight": 2446,
  "BestChainHash": "fb749177c2f43db8c7d73ea050240b9f870c40584f044b13e7ec146c460b0eff",
  "BestChainHeight": 2449
}
```

4.3 Be a candidate node

Full nodes need to call Election contract to become candidate nodes. The nodes need to mortgage 10W ELF to participate in the election, please make sure that the account of the nodes has enough tokens.

To facilitate the quick demonstration, we directly transfer the token from the first initial node account to the full node account:

```
aelf-command send AElf.ContractNames.Token Transfer '{"symbol": "ELF", "to":
↪"Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC", "amount": "20000000000000"}'
```

By checking the balance of the full node account, we can see that the full node account has enough tokens, 20W ELF:

```
aelf-command call AElf.ContractNames.Token GetBalance '{"symbol": "ELF", "owner":
↪"Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC"}'

Result:
{
  "symbol": "ELF",
  "owner": "Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC",
  "balance": "20000000000000"
}
```

Full node announces election with admin specified in params:

```
aelf-command send AElf.ContractNames.Election AnnounceElection '{"value":
↪"Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC"}' -a␣
↪Q3t34SAEsxAQrSQidTRzDonWNTPpSTgH8bqu8pQUGCSWRPdRC
```

By inquiring candidate information, we can see the full node is already candidates:

```
aelf-command call AElf.ContractNames.Election GetCandidateInformation '{"value":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9a
↪"}'

Result:
{
  "terms": [],
  "pubkey":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9a
↪",
  "producedBlocks": "0",
  "missedTimeSlots": "0",
  "continualAppointmentCount": "0",
  "announcementTransactionId":
↪"8cc8eb5de35e390e4f7964bbdc7edc433498b041647761361903c6165b9f8659",
  "isCurrentCandidate": true
}
```

4.4 User vote election

For the simulated user voting scenario, we create a user account:

```
aelf-command create
```

(continues on next page)

(continued from previous page)

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic             : walnut market museum play grunt chuckle hybrid␣
↪accuse relief misery share meadow
AElf [Info]: Private Key          :␣
↪919a220fac2d80e674a256f2367ac840845f344269f4dcdd56d37460de17f947
AElf [Info]: Public Key           :␣
↪04794948de40ffda2a6c884d7e6a99bb8e42b8b96b9ee5cc4545da3a1d5f7725eec93de62ddbfb598ef6f04fe52aa310acc
AElf [Info]: Address              : ZBBPU7DMVQ72YBQNmaKTDPKaAkHNzzA3naH5B6kE7cBm8g1ei
```

After the user account is created successfully, we will first trsnfer some tokens to the account for voting.

```
aelf-command send AElf.ContractNames.Token Transfer '{"symbol": "ELF", "to":
↪"ZBBPU7DMVQ72YBQNmaKTDPKaAkHNzzA3naH5B6kE7cBm8g1ei", "amount": "200000000000"}'
```

Confirm the tokens has been received:

```
aelf-command call AElf.ContractNames.Token GetBalance '{"symbol": "ELF", "owner":
↪"ZBBPU7DMVQ72YBQNmaKTDPKaAkHNzzA3naH5B6kE7cBm8g1ei"}'

Result:
{
  "symbol": "ELF",
  "owner": "ZBBPU7DMVQ72YBQNmaKTDPKaAkHNzzA3naH5B6kE7cBm8g1ei",
  "balance": "200000000000"
}
```

Users vote on candidate nodes through the election contract.

```
aelf-command send AElf.ContractNames.Election Vote '{"candidatePubkey":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9a
↪","amount":2000000000,"endTimestamp":{"seconds":1600271999,"nanos":999000}}' -a␣
↪ZBBPU7DMVQ72YBQNmaKTDPKaAkHNzzA3naH5B6kE7cBm8g1ei
```

By inquiring the votes of candidates, we can see that the full node has successfully obtained 20 votes.

```
aelf-command call AElf.ContractNames.Election GetCandidateVote '{"value":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9a
↪"}'

Result:
{
  "obtainedActiveVotingRecordIds": [
    "172375e9cee303ce60361aa73d7326920706553e80f4485f97ffefdb904486f1"
  ],
  "obtainedWithdrawnVotingRecordIds": [],
  "obtainedActiveVotingRecords": [],
  "obtainedWithdrawnVotesRecords": [],
  "obtainedActiveVotedVotesAmount": "2000000000",
  "allObtainedVotedVotesAmount": "2000000000",
  "pubkey":
↪"BO7QDrAJzNKDeY44Yngc69Je1qRkHg4bfQ47a1kCUEBnn8TcDtyd4Wa9YwxyVRiKmurfyDL9rggoJw93xu8meQU=
↪"
}
```

4.5 Become production node

At the next election, the candidate nodes with votes in the first 17 are automatically elected as production nodes, and the current production node list can be viewed through consensus contracts.

Quantity 17 is the default maximum production node quantity, which can be modified by proposal. Please refer to the Consensus and Proposal Contract API for details.

```
aelf-command call AElf.ContractNames.Consensus GetCurrentMinerPubkeyList '{}'

Result:
{
  "pubkeys": [

↪"0499d3bb14337961c4d338b9729f46b20de8a49ed38e260a5c19a18da569462b44b820e206df8e848185dac6c139f0539
↪",

↪"048397dfd9e1035fdd7260329d9492d88824f42917c156aef93fd7c2e3ab73b636f482b8ceb5cb435c556bfa067445a86
↪",

↪"041cc962a51e7bbdd829a8855eca8a03fda708fdf31969251321cb31edadd564bf3c6e7ab31b4c1f49f0f206be81dbe68
↪",

↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9
↪"
  ]
}
```

4.6 Add more production nodes

Repeat steps 2-4 to add more production nodes. When the number of initial nodes plus the number of candidate nodes exceeds the maximum number of production node, the replacement will replace the initial nodes step by step, and the replaced initial nodes are not allowed to run for election again. At this time, the initial node has completed its responsibility of starting AElf Blockchain.

CHAPTER 5

How to join the testnet

There's two ways to run a AElf node: you can either use Docker (recommended method) or run the binaries available on Github. Before you jump into the guides and tutorials you'll need to install the following tools and frameworks. For most of these dependencies we provide ready-to-use command line instructions. In case of problems or if you have more complex needs, we provide more information in the *Environment setup* section.

Summary of the steps to set up a node:

1. Execute the snapshot download script and load the snapshot into the database.
2. Download our template setting files and docker run script.
3. Modify the appsettings according to your needs.
4. Run and check the node.

Hardware suggestion: for the AElf testnet we use the following Amazon configuration: c5.large instance with 2 vCPUs, 4GiB RAM and a 200GiB hard drive for each node we run. We recommend using something similar per node that you want to run (one for the mainchain node and one per side chain node).

Note: any server you use to run a node should be time synced via NTP. Failing to do this will prevent your node from syncing.

5.1 Setup the database

We currently support two key-value databases to store our nodes data: Redis and SSDB, but for the testnet we only provide snapshots for SSDB. We will configure two SSDB instances, one for chain database and one for the state database (run these on different machines for better performances).

5.1.1 Import the snapshot data

After you've finished setting up the database, download the latest snapshots. The following gives you the template for the download URL,but you have to specify the snapshot date. We recommend you get the latest.

Restore the chain database from snapshot:

```
>> mkdir snapshot
>> cd snapshot

## fetch the snapshot download script
>> curl -O -s https://aelf-node.s3-ap-southeast-1.amazonaws.com/snapshot/testnet/
↪download-mainchain-db.sh

## execute the script, you can optionally specify a date by appending "yyyymmdd" as␣
↪parameter
>> sh download-mainchain-db.sh

## chain database: decompress and load the chain database snapshot
>> tar xvzf aelf-testnet-mainchain-chaindb-*.tar.gz
>> stop your chain database instance (ssdb server)
>> cp -r aelf-testnet-mainchain-chaindb-*/* /path/to/install/chaindb/ssdb/var/
>> start your chain database instance
>> enter ssdb console (ssdb-cli) use the "info" command to confirm that the data has␣
↪been imported)

## state database : decompress and load the state database
>> tar xvzf aelf-testnet-mainchain-statedb-*.tar.gz
>> stop your state database instance (ssdb server)
>> cp -r aelf-testnet-mainchain-statedb-*/* /path/to/install/statedb/ssdb/var/
>> start your state database instance
>> enter ssdb console (ssdb-cli) use the "info" command to confirm that the data has␣
↪been imported)
```

5.2 Node configuration

5.2.1 Generating the nodes account

This section explains how to generate an account for the node. First you need to install the aelf-command npm package. Open a terminal and enter the following command to install aelf-command:

```
>> npm i -g aelf-command
```

After installing the package, you can use the following command to create an account/key-pair:

```
>> aelf-command create
```

The command prompts for a password, enter it and don't forget it. The output of the command should look something like this:

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : term jar tourist monitor melody tourist catch sad␣
↪ankle disagree great adult
AElf [Info]: Private Key         :␣
↪34192c729751bd6ac0a5f18926d74255112464b471aec499064d5d1e5b8ff3ce
AElf [Info]: Public Key          :␣
↪04904e51a944ab13b031cb4fead8caa6c027b09661dc5550ee258ef5c5e78d949b1082636dc8e27f20bc427b25b99a1cada
AElf [Info]: Address             : 29KM437eJRRuTfvhsB8QAsyVvi8mmyN9Wqqame6TsJhrqXbeWd
? Save account info into a file? Yes
? Enter a password: *********
```

(continues on next page)

(continued from previous page)

```
? Confirm password: *********
✓ Account info has been saved to "/usr/local/share/aelf/keys/
↪29KM437eJRRuTfvhsB8QAsyVvi8mmyN9Wqqame6TsJhrqXbeWd.json"
```

In the next steps of the tutorial you will need the Public Key and the Address for the account you just created. You'll notice the last line of the commands output will show you the path to the newly created key. The aelf directory is the data directory (datadir) and this is where the node will read the keys from.

Note that a more detailed section about the cli can be found *command line interface*.

5.2.2 Prepare node configuration

```
## download the settings template and docker script
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0-rc1/
↪aelf-testnet-mainchain.zip
>> unzip aelf-testnet-mainchain.zip
>> mv aelf-testnet-mainchain /opt/aelf-node
```

Update the appsetting.json file with your account. This will require the information printed during the creation of the account. Open the appsettings.json file and edit the following sections.

The account/key-pair associated with the node we are going to run:

```
{
    "Account": {
        "NodeAccount": "2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H",
        "NodeAccountPassword": "********"
    }
}
```

You also have to configure the database connection strings (port/db number):

```
{
    "ConnectionStrings": {
        "BlockchainDb": "redis://your chain database server ip address:port",
        "StateDb": "redis://your state database server ip address:port"
    },
}
```

If you use docker to run the node and it is on the same server as the database, please do not use 127.0.0.1 as the database monitoring ip.

Next add the testnet mainchain nodes as peer (bootnode peers):

```
{
    "Network": {
        "BootNodes": [
            "xxx.xxxx.xxx.xxx:6800",
            "..."
        ],
        "ListeningPort": 6800
    }
}
```

Note: if your infrastructure is behind a firewall you need to open the P2P listening port of the node. You also need to configure your listening ip and port for the side chain connections in `appsettings.MainChain.TestNet.json`:

```
{
    "CrossChain": {
        "Grpc": {
            "LocalServerPort": 5000,
        }
    },
}
```

5.3 Running a full node with Docker

To run the node with Docker, enter the following commands:

```
## pull AElf's image and navigate to the template folder to execute the start script
>> docker pull aelf/node:testnet-v1.0.0
>> cd /opt/aelf-node
>> sh aelf-node.sh start aelf/node:testnet-v1.0.0
```

to stop the node you can run:

```
>> sh aelf-node.sh stop
```

5.4 Running a full node with the binary release

Most of AElf is developed with dotnet core, so to run the binaries you will need to download and install the .NET Core SDK before you start: Download .NET Core 6.0. For now AElf depends on version 6.0 of the SDK, on the provided link find the download for your platform, and install it.

Get the latest release with the following commands:

```
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0-rc1/
↪aelf.zip
>> unzip aelf.zip
>> mv aelf /opt/aelf-node/
```

Enter the configuration folder and run the node:

```
>> cd /opt/aelf-node
>> dotnet aelf/AElf.Launcher.dll
```

5.5 Running a full node with the source

The most convenient way is to directly use docker or the binary packages, but if you want you can compile from source code. First make sure the code version is consistent (current is release AELF v1.0.0), and secondly make sure to compile on a Ubuntu Linux machine (we recommend Ubuntu 18.04.2 LTS) and have dotnet core SDK version 6.0 installed. This is because different platforms or compilers will cause the dll hashes to be inconsistent with the current chain.

5.6 Check the node

You now should have a node that's running, to check this run the following command that will query the node for its current block height:

```
aelf-command get-blk-height -e http://your node ip address:port
```

5.7 Run side-chains

This section explains how to set up a side-chain node, you will have to repeat these steps for all side chains (currently only one is running):

1. Fetch the appsettings and the docker run script.
2. Download and restore the snapshot data with the URLs provided below (steps are the same as in A - Setup the database).
3. Run the side-chain node.

Running a side chain is very much like running a mainchain node, only configuration will change. Here you can find the instructions for sidechain1:

```
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0-rc1/
↪aelf-testnet-sidechain1.zip
>> unzip aelf-testnet-sidechain1.zip
>> mv aelf-testnet-sidechain1 /opt/aelf-node
```

In order for a sidechain to connect to a mainchain node you need to modify the `appsettings.SideChain.TestNet.json` with your node information.

```
{
    "CrossChain": {
        "Grpc": {
            "ParentChainServerPort": 5000,
            "ParentChainServerIp": "your mainchain ip address",
            "ListeningPort": 5001,
        },
        "ParentChainId": "AELF"
    }
}
```

Here you can find the snapshot data for the only current side-chain running, optionally you can specify the date, but we recommend you get the latest:

```
>> curl -O -s https://aelf-node.s3-ap-southeast-1.amazonaws.com/snapshot/testnet/
↪download-sidechain1-db.sh
```

Here you can find the list of templates folders (appsettings and docker run script) for the side-chain:

```
wget https://github.com/AElfProject/AElf/releases/download/v1.0.0-rc1/aelf-testnet-
↪sidechain1.zip
```

Each side chain has its own P2P network, add the testnet sidechain nodes as peer:

```
bootnode → ["xxx.xxxx.xxx.xxx:6800", "..."]
```

```
{
    "Network": {
        "BootNodes": [
            "Add the right boot node according sidechain"
        ],
        "ListeningPort": 6800
    }
}
```

CHAPTER 6

How to join the mainnet

There's two ways to run a AElf node: you can either use Docker (recommended method) or run the binaries available on Github. Before you jump into the guides and tutorials you'll need to install the following tools and frameworks. For most of these dependencies we provide ready-to-use command line instructions. In case of problems or if you have more complex needs, we provide more information in the *Environment setup* section.

Summary of the steps to set up a node:

1. Execute the snapshot download script and load the snapshot into the database.
2. Download our template setting files and docker run script.
3. Modify the appsettings according to your needs.
4. Run and check the node.

Hardware suggestion: for the AElf mainnet we use the following Amazon configuration: c5.xlarge instance with 4 vCPUs, 8GiB RAM and a 500GiB hard drive for each node we run. We recommend using something similar per node that you want to run (one for the mainchain node and one per side chain node).

Note: any server you use to run a node should be time synced via NTP. Failing to do this will prevent your node from syncing.

6.1 Setup the database

We currently support two key-value databases to store our nodes data: Redis and SSDB, but for the mainnet we only provide snapshots for SSDB. We will configure two SSDB instances, one for chain database and one for the state database (run these on different machines for better performances).

6.1.1 Import the snapshot data

After you've finished setting up the database, download the latest snapshots. The following gives you the template for the download URL,but you have to specify the snapshot date. We recommend you get the latest.

Restore the chain database from snapshot:

```
>> mkdir snapshot
>> cd snapshot

## fetch the snapshot download script
>> curl -O -s https://aelf-backup.s3.ap-northeast-2.amazonaws.com/snapshot/mainnet/
↪download-mainchain-db.sh

## execute the script, you can optionally specify a date by appending "yyyymmdd" as␣
↪parameter
>> sh download-mainchain-db.sh

## chain database: decompress and load the chain database snapshot
>> tar xvzf aelf-mainnet-mainchain-chaindb-*.tar.gz
>> stop your chain database instance (ssdb server)
>> cp -r aelf-mainnet-mainchain-chaindb-*/* /path/to/install/chaindb/ssdb/var/
>> start your chain database instance
>> enter ssdb console (ssdb-cli) use the "info" command to confirm that the data has␣
↪been imported)

## state database : decompress and load the state database
>> tar xvzf aelf-mainnet-mainchain-statedb-*.tar.gz
>> stop your state database instance (ssdb server)
>> cp -r aelf-mainnet-mainchain-statedb-*/* /path/to/install/statedb/ssdb/var/
>> start your state database instance
>> enter ssdb console (ssdb-cli) use the "info" command to confirm that the data has␣
↪been imported)
```

6.2 Node configuration

6.2.1 Generating the nodes account

This section explains how to generate an account for the node. First you need to install the aelf-command npm package. Open a terminal and enter the following command to install aelf-command:

```
>> npm i -g aelf-command
```

After installing the package, you can use the following command to create an account/key-pair:

```
>> aelf-command create
```

The command prompts for a password, enter it and don't forget it. The output of the command should look something like this:

```
AElf [Info]: Your wallet info is :
AElf [Info]: Mnemonic            : term jar tourist monitor melody tourist catch sad␣
↪ankle disagree great adult
AElf [Info]: Private Key         :␣
↪34192c729751bd6ac0a5f18926d74255112464b471aec499064d5d1e5b8ff3ce
AElf [Info]: Public Key          :␣
↪04904e51a944ab13b031cb4fead8caa6c027b09661dc5550ee258ef5c5e78d949b1082636dc8e27f20bc427b25b99a1cada
AElf [Info]: Address             : 29KM437eJRRuTfvhsB8QAsyVvi8mmyN9Wqqame6TsJhrqXbeWd
? Save account info into a file? Yes
? Enter a password: *********
```

(continues on next page)

(continued from previous page)

```
? Confirm password: *********
✓ Account info has been saved to "/usr/local/share/aelf/keys/
↪29KM437eJRRuTfvhsB8QAsyVvi8mmyN9Wqqame6TsJhrqXbeWd.json"
```

In the next steps of the tutorial you will need the Public Key and the Address for the account you just created. You'll notice the last line of the commands output will show you the path to the newly created key. The aelf directory is the data directory (datadir) and this is where the node will read the keys from.

Note that a more detailed section about the cli can be found *command line interface*.

6.2.2 Prepare node configuration

```
## download the settings template and docker script
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0/aelf-
↪mainnet-mainchain.zip
>> unzip aelf-mainnet-mainchain.zip
>> mv aelf-mainnet-mainchain /opt/aelf-node
```

Update the appsetting.json file with your account. This will require the information printed during the creation of the account. Open the appsettings.json file and edit the following sections.

The account/key-pair associated with the node we are going to run:

```
{
    "Account": {
        "NodeAccount": "2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H",
        "NodeAccountPassword": "********"
    }
}
```

You also have to configure the database connection strings (port/db number):

```
{
    "ConnectionStrings": {
        "BlockchainDb": "redis://your chain database server ip address:port",
        "StateDb": "redis://your state database server ip address:port"
    },
}
```

If you use docker to run the node and it is on the same server as the database, please do not use 127.0.0.1 as the database monitoring ip.

Next add the mainnet mainchain nodes as peer (bootnode peers):

```
{
    "Network": {
        "BootNodes": [
            "xxx.xxxx.xxx.xxx:6800",
            "..."
        ],
        "ListeningPort": 6800
    }
}
```

Note: if your infrastructure is behind a firewall you need to open the P2P listening port of the node. You also need to configure your listening ip and port for the side chain connections in `appsettings.MainChain.MainNet.json`:

```
{
    "CrossChain": {
        "Grpc": {
            "LocalServerPort": 5000,
        }
    },
}
```

6.3 Running a full node with Docker

To run the node with Docker, enter the following commands:

```
## pull AElf's image and navigate to the template folder to execute the start script
>> docker pull aelf/node:mainnet-v1.0.0
>> cd /opt/aelf-node
>> sh aelf-node.sh start aelf/node:mainnet-v1.0.0
```

to stop the node you can run:

```
>> sh aelf-node.sh stop
```

6.4 Running a full node with the binary release

Most of AElf is developed with dotnet core, so to run the binaries you will need to download and install the .NET Core SDK before you start: Download .NET Core 6.0. For now AElf depends on version 6.0 of the SDK, on the provided link find the download for your platform, and install it.

Get the latest release with the following commands:

```
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0/aelf.
↪zip
>> unzip aelf.zip
>> mv aelf /opt/aelf-node/
```

Enter the configuration folder and run the node:

```
>> cd /opt/aelf-node
>> dotnet aelf/AElf.Launcher.dll
```

6.5 Running a full node with the source

The most convenient way is to directly use docker or the binary packages, but if you want you can compile from source code. First make sure the code version is consistent (current is release AELF v1.0.0), and secondly make sure to compile on a Ubuntu Linux machine (we recommend Ubuntu 18.04.2 LTS) and have dotnet core SDK version 6.0 installed. This is because different platforms or compilers will cause the dll hashes to be inconsistent with the current chain.

6.6 Check the node

You now should have a node that's running, to check this run the following command that will query the node for its current block height:

```
aelf-command get-blk-height -e http://your node ip address:port
```

6.7 Run side-chains

This section explains how to set up a side-chain node, you will have to repeat these steps for all side chains (currently only one is running):

1. Fetch the appsettings and the docker run script.
2. Download and restore the snapshot data with the URLs provided below (steps are the same as in Setup the database).
3. Run the side-chain node.

Running a side chain is very much like running a mainchain node, only configuration will change. Here you can find the instructions for sidechain1:

```
>> cd /tmp/ && wget https://github.com/AElfProject/AElf/releases/download/v1.0.0/aelf-
↪mainnet-sidechain1.zip
>> unzip aelf-mainnet-sidechain1.zip
>> mv aelf-mainnet-sidechain1 /opt/aelf-node
```

In order for a sidechain to connect to a mainchain node you need to modify the `appsettings.SideChain.MainNet.json` with your node information.

```
{
    "CrossChain": {
        "Grpc": {
            "ParentChainServerPort": 5001,
            "ParentChainServerIp": "your mainchain ip address",
            "ListeningPort": 5011,
        },
        "ParentChainId": "AELF",
        "Economic": {
            "SymbolListToPayTxFee": "WRITE,READ,STORAGE,TRAFFIC",
            "SymbolListToPayRental": "CPU,RAM,DISK,NET"
    }
  }
}
```

Here you can find the snapshot data for the only current side-chain running, optionally you can specify the date, but we recommend you get the latest:

```
>> curl -O -s https://aelf-backup.s3.ap-northeast-2.amazonaws.com/snapshot/mainnet/
↪download-sidechain-db.sh
```

Here you can find the list of templates folders (appsettings and docker run script) for the side-chain:

```
wget https://github.com/AElfProject/AElf/releases/download/v1.0.0/aelf-mainnet-
↪sidechain1.zip
```

Each side chain has its own P2P network, add the mainnet sidechain nodes as peer:

```
bootnode → ["xxx.xxxx.xxx.xxx:6800", "..."]
```

```
{
    "Network": {
        "BootNodes": [
            "Add the right boot node according sidechain"
        ],
        "ListeningPort": 6800
    }
}
```

CHAPTER 7

Running a side chain

7.1 Requesting the creation of a side chain

Side chains can be created in the AELF ecosystem to enable scalability. This part is going to introduce these periods in detail.

7.1.1 Side chain creation api

Anyone can request the side chain creation in the AELF ecosystem. The proposer/creator of a new side chain will need to request the creation of the side chain through the cross-chain contract on the main-chain. The request contains different fields that will determine the type of side chain that will be created.

This section show the API to use in order to propose the creation of a side chain. The fields that are in the `SideChainCreationRequest` will determine the type of side chain that is created. For more api details, you can follow the `RequestSideChainCreation` in *Crosschain contract*.

A new proposal about the side chain creation would be created and the event `ProposalCreated` containing proposal id would be fired. A parliament organization which is specified since the chain launched is going to approve this proposal in 24 hours(refer to *Parliament contract* for detail). Proposer is able to release the side chain creation request with proposal id once the proposal can be released. Refer `ReleaseSideChainCreation` in *Crosschain contract*.

New side chain would be created and the event `SideChainCreatedEvent` containing chain id would be fired.

Side chain node can be launched since it is already created on main chain. Side chain id from the creation result should be configured correctly before launching the side chain node. Please make sure cross chain communication context is correctly set, because side chain node is going to request main chain node for chain initialization data. For more details, check *side chain node running* tutorial.

7.1.2 Side chain types

Two types of side-chain's currently exist: **exclusive** or **shared**. An **exclusive** side-chain is a type of dedicated side-chain (as opposed to shared) that allows developers to choose the transaction fee model and set the transaction fee price. The creator has exclusive use of this side-chain. For example, only creator of this **exclusive** side-chain can propose to deploy a new contract.

7.1.3 Pay for Side chain

Indexing fee

Indexing fee, literally, is paid for the side chain indexing. You can specify the indexing fee price and prepayments amount when you request side chain creation. *Cross chain contract* is going to charge your prepayments once the side chain created and pay the miner who indexes the side chain block every time.

Resource fee

Developers of an exclusive side-chain pay the producers for running it by paying CPU, RAM, DISK, NET resource tokens: this model is called *charge-by-time*. The amount side chain creator must share with the producers is set after creation of the chain. The **exclusive** side-chain is priced according to the time used. The unit price of the fee is determined through negotiation between the production node and the developer.

See Economic whitepaper - 4.3 Sidechain Developer Charging Model for more information.

7.1.4 Simple demo for side chain creation request

When a user (usually a developer) feels the need to create a new side chain on AElf he must call the cross-chain contract and request a side chain creation. After requested, parliament organization members will either approve this creation or reject it. If the request is approved, the developer must then release the proposal.

Throughout this tutorial we'll give step-by-step code snippets that use the aelf-js-sdk to create a new side chain, the full script will be given at the end of the tutorial.

This creation of a side chain (logical, on-chain creation) is done in four steps:

- the developer must *allow/approve* some tokens to the cross-chain contract of the main chain.
- the developer calls the cross-chain contract of the main chain, to *request* the creation.
- the parliament organization members must *approve* this request.
- finally the developer must *release* the request to finalize the creation.

Keep in mind that this is just the logical on-chain creation of the side chain. After the side chain is released there's extra steps needed for it to be a fully functional blockchain, including the producers running the side chain's nodes.

Set-up

If you want to test the creation process you will need a producer node running and the following:

- you need a key-pair (account) created, this will be your Producer (in this tutorial we also use the producer to create the creation request).
- the node needs to be configured with an API endpoint, account and miner list that correspond to what is in the script.

The following snippet shows constants and initialization code used in the script:

```
const AElf = require('aelf-sdk');
const Wallet = AElf.wallet;

const { sha256 } = AElf.utils;

// set the private key of the block producer.
// REPLACE
const defaultPrivateKey =
↪'e119487fea0658badc42f089fbaa56de23d8c0e8d999c5f76ac12ad8ae897d76';
const defaultPrivateKeyAddress = 'HEtBQStfqu53cHVC3PxJU6iGP3RGxiNUfQGvAPTjfrF3ZWH3U';

// load the wallet associated with your block producers account.
const wallet = Wallet.getWalletByPrivateKey(defaultPrivateKey);

// API link to the node
// REPLACE
const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:1234'));

// names of the contracts that will be used.
const tokenContractName = 'AElf.ContractNames.Token';
const parliamentContractName = 'AElf.ContractNames.Parliament';
const crossChainContractName = 'AElf.ContractNames.CrossChain';

...

const createSideChain = async () => {
    // check the chain status to make sure the node is running
    const chainStatus = await aelf.chain.getChainStatus({sync: true});
    const genesisContract = await aelf.chain.contractAt(chainStatus.
↪GenesisContractAddress, wallet)
        .catch((err) => {
        console.log(err);
        });

    // get the addresses of the contracts that we'll need to call
    const tokenContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(tokenContractName));
    const parliamentContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(parliamentContractName));
    const crossChainContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(crossChainContractName));

    // build the aelf-sdk contract instance objects
    const parliamentContract = await aelf.chain.contractAt(parliamentContractAddress,␣
↪wallet);
    const tokenContract = await aelf.chain.contractAt(tokenContractAddress, wallet);
    const crossChainContract = await aelf.chain.contractAt(crossChainContractAddress,␣
↪wallet);

    ...
}
```

When running the script, the **createSideChain** will be executed and automatically will run through the full process of creating the side chain.

Creation of the side chain

Set the Allowance.

First the developer must approve some ELF tokens for use by the cross-chain contract.

```
var setAllowance = async function(tokenContract, crossChainContractAddress)
{
    // set some allowance to the cross-chain contract
    const approvalResult = await tokenContract.Approve({
        symbol:'ELF',
        spender: crossChainContractAddress,
        amount: 20000
        });

    let approveTransactionResult = await pollMining(approvalResult.TransactionId);
}
```

Creation request

In order to request a side chain creation the developer must call **RequestSideChainCreation** on the cross-chain contract, this will create a proposal with the **Parliament** contract. After calling this method, a **ProposalCreated** log will be created in which the **ProposalId** be found. This ID will enable the producers to approve it.

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

message SideChainCreationRequest {
    // The cross chain indexing price.
    int64 indexing_price = 1;
    // Initial locked balance for a new side chain.
    int64 locked_token_amount = 2;
    // Creator privilege boolean flag: True if chain creator privilege preserved,␣
↪otherwise false.
    bool is_privilege_preserved = 3;
    // Side chain token information.
    SideChainTokenCreationRequest side_chain_token_creation_request = 4;
    // A list of accounts and amounts that will be issued when the chain starts.
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 5;
    // The initial rent resources.
    map<string, int32> initial_resource_amount = 6;
}

message SideChainTokenCreationRequest{
    // Token symbol of the side chain to be created
    string side_chain_token_symbol = 1;
    // Token name of the side chain to be created
    string side_chain_token_name = 2;
    // Token total supply of the side chain to be created
    int64 side_chain_token_total_supply = 3;
    // Token decimals of the side chain to be created
    int32 side_chain_token_decimals = 4;
}

message SideChainTokenInitialIssue{
    // The account that will be issued.
    aelf.Address address = 1;
```

(continues on next page)

(continued from previous page)

```
    // The amount that will be issued.
    int64 amount = 2;
}
```

In order for the creation request to succeed, some assertions must pass:

- the Sender can only have one pending request at any time.
- the locked_token_amount cannot be lower than the indexing price.
- if **is_privilege_preserved** is true, which means it requests **exclusive** side chain, the token initial issue list cannot be empty and all with an **amount** greater than 0.
- if **is_privilege_preserved** is true, which means it requests **exclusive** side chain, the **initial_resource_amount** must contain all resource tokens of the chain and the value must be greater than 0.
- the allowance approved to cross chain contract from the proposer (Sender of the transaction) cannot be lower than the **locked_token_amount**.
- no need to provide data about side chain token if **is_privilege_preserved** is false, and side chain token won't be created even you provide token info.

```
 const sideChainCreationRequestTx = await crossChainContract.RequestSideChainCreation(
↪{
    indexingPrice: 1,
    lockedTokenAmount: '20000',
    isPrivilegePreserved: true,
    sideChainTokenCreationRequest: {
        sideChainTokenDecimals: 8,
        sideChainTokenName: 'SCATokenName',
        sideChainTokenSymbol: 'SCA',
        sideChainTokenTotalSupply: '100000000000000000',
    },
    sideChainTokenInitialIssueList: [
        {
            address: '28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK',
            amount: '1000000000000000'
        }
    ],
    initialResourceAmount: { CPU: 2, RAM: 4, DISK: 512, NET: 1024 },
});

 let sideChainCreationRequestTxResult = await pollMining(sideChainCreationRequestTx.
↪TransactionId);

 // deserialize the log to get the proposal's ID.
 let deserializedLogs = parliamentContract.
↪deserializeLog(sideChainCreationRequestTxResult.Logs, 'ProposalCreated');
```

The last line will print the proposal ID and this is what will be used for approving by the producers.

Approval from producers

This is where the parliament organization members approve the proposal:

```
var proposalApproveTx = await parliamentContract.Approve(deserializedLogs[0].
↪proposalId);
await pollMining(proposalApproveTx.TransactionId);
```

Note: when calling **Approve** it will be the *Sender* of the transaction that approves. Here the script is set to use the key of one parliament organization member, see full script at the end.

Release

This part of the script releases the proposal:

```
var releaseResult = await crossChainContract.ReleaseSideChainCreation({
    proposalId: deserializedLogs[0].proposalId
});

let releaseTxResult = await pollMining(releaseResult.TransactionId);

// Parse the logs to get the chain id.
let sideChainCreationEvent = crossChainContract.deserializeLog(releaseTxResult.Logs,
↪'SideChainCreatedEvent');
```

This is the last step involved in creating a side chain, after this the chain id of the new side chain is accessible in the **SideChainCreatedEvent** event log.

Full script

This section presents the full script. Remember that in order to run successfully, a node must be running, configured with one producer. The configured producer must match the **defaultPrivateKey** and **defaultPrivateKeyAddress** of the script.

Also, notice that this script by default tries to connect to the node's API at the following address http://127.0.0.1:1234, if your node is listening on a different address you have to modify the address.

If you haven't already installed it, you need the aelf-sdk:

```
npm install aelf-sdk
```

You can simply run the script from anywhere:

```
node sideChainProposal.js
```

sideChainProposal.js:

```
const AElf = require('aelf-sdk');
const Wallet = AElf.wallet;

const { sha256 } = AElf.utils;

// set the private key of the block producer
const defaultPrivateKey =
↪'e119487fea0658badc42f089fbaa56de23d8c0e8d999c5f76ac12ad8ae897d76';
const defaultPrivateKeyAddress = 'HEtBQStfqu53cHVC3PxJU6iGP3RGxiNUfQGvAPTjfrF3ZWH3U';

const wallet = Wallet.getWalletByPrivateKey(defaultPrivateKey);
```

(continues on next page)

(continued from previous page)

```
// link to the node
const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:8000'));

if (!aelf.isConnected()) {
    console.log('Could not connect to the node.');
}

const tokenContractName = 'AElf.ContractNames.Token';
const parliamentContractName = 'AElf.ContractNames.Parliament';
const crossChainContractName = 'AElf.ContractNames.CrossChain';

var pollMining = async function(transactionId) {
    console.log(`>> Waiting for ${transactionId} the transaction to be mined.`);

    for (i = 0; i < 10; i++) {
        const currentResult = await aelf.chain.getTxResult(transactionId);
        // console.log('transaction status: ' + currentResult.Status);

        if (currentResult.Status === 'MINED')
            return currentResult;

        await new Promise(resolve => setTimeout(resolve, 2000))
            .catch(function () {
                console.log("Promise Rejected");
            });;
    }
}

var setAllowance = async function(tokenContract, crossChainContractAddress)
{
    console.log('\n>>>> Setting allowance for the cross-chain contract.');

    // set some allowance to the cross-chain contract
    const approvalResult = await tokenContract.Approve({
        symbol:'ELF',
        spender: crossChainContractAddress,
        amount: 20000
    });

    await pollMining(approvalResult.TransactionId);
}

var checkAllowance = async function(tokenContract, owner, spender)
{
    console.log('\n>>>> Checking the cross-chain contract\'s allowance');

    const checkAllowanceTx = await tokenContract.GetAllowance.call({
        symbol: 'ELF',
        owner: owner,
        spender: spender
    });

    console.log(`>> allowance to the cross-chain contract: ${checkAllowanceTx.
↪allowance} ${checkAllowanceTx.symbol}`);
}

const createSideChain = async () => {
```

(continues on next page)

(continued from previous page)

```
    // get the status of the chain in order to get the genesis contract address
    console.log('Starting side chain creation script\n');

    const chainStatus = await aelf.chain.getChainStatus({sync: true});
    const genesisContract = await aelf.chain.contractAt(chainStatus.
↪GenesisContractAddress, wallet)
        .catch((err) => {
            console.log(err);
        });

    // get the addresses of the contracts that we'll need to call
    const tokenContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(tokenContractName));
    const parliamentContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(parliamentContractName));
    const crossChainContractAddress = await genesisContract.GetContractAddressByName.
↪call(sha256(crossChainContractName));

    // build the aelf-sdk contract object
    const parliamentContract = await aelf.chain.contractAt(parliamentContractAddress,␣
↪wallet);
    const tokenContract = await aelf.chain.contractAt(tokenContractAddress, wallet);
    const crossChainContract = await aelf.chain.contractAt(crossChainContractAddress,␣
↪wallet);


    // 1. set and check the allowance, spender is the cross-chain contract
    await setAllowance(tokenContract, crossChainContractAddress);
    await checkAllowance(tokenContract, defaultPrivateKeyAddress,␣
↪crossChainContractAddress);

    // 2. request the creation of the side chain with the cross=chain contract
    console.log('\n>>>> Requesting the side chain creation.');
    const sideChainCreationRequestTx = await crossChainContract.
↪RequestSideChainCreation({
        indexingPrice: 1,
        lockedTokenAmount: '20000',
        isPrivilegePreserved: true,
        sideChainTokenCreationRequest: {
            sideChainTokenDecimals: 8,
            sideChainTokenName: 'SCATokenName',
            sideChainTokenSymbol: 'SCA',
            sideChainTokenTotalSupply: '100000000000000000',
        },
        sideChainTokenInitialIssueList: [
            {
                address: '28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK',
                amount: '1000000000000000'
            }
        ],
        initialResourceAmount: { CPU: 2, RAM: 4, DISK: 512, NET: 1024 },
    });

    let sideChainCreationRequestTxResult = await␣
↪pollMining(sideChainCreationRequestTx.TransactionId);
```

(continues on next page)

(continued from previous page)

```
    // deserialize the log to get the proposal's ID.
    let deserializedLogs = parliamentContract.
↪deserializeLog(sideChainCreationRequestTxResult.Logs, 'ProposalCreated');
    console.log(`>> side chain creation request proposal id ${JSON.
↪stringify(deserializedLogs[0].proposalId)}`);

    // 3. Approve the proposal
    console.log('\n>>>> Approving the proposal.');

    var proposalApproveTx = await parliamentContract.Approve(deserializedLogs[0].
↪proposalId);
    await pollMining(proposalApproveTx.TransactionId);

    // 3. Release the side chain
    console.log('\n>>>> Release the side chain.');

    var releaseResult = await crossChainContract.ReleaseSideChainCreation({
        proposalId: deserializedLogs[0].proposalId
    });

    let releaseTxResult = await pollMining(releaseResult.TransactionId);

    // Parse the logs to get the chain id.
    let sideChainCreationEvent = crossChainContract.deserializeLog(releaseTxResult.
↪Logs, 'SideChainCreatedEvent');
    console.log('Chain chain created : ');
    console.log(sideChainCreationEvent);
};

createSideChain().then(() => {console.log('Done.')});
```

7.2 Running a side chain (after its release)

This tutorial will explain how to run a side chain node after it has been *approved* by the producers and *released* by the creator. After the creation of the side chain, the producers need to run a side chain node.

A side chain node is usually very similar to a main-chain node because both are based on AElf software and have common modules. The main difference is the configuration which varies depending on if the node is a side chain or not.

Note: this tutorial assumes the following:

- you already have a main-chain node running.
- the creation of the side chain has already been approved and released.

It's also **important** to know that the key-pair (account) used for mining on the side chain must be the **same** as the one you use for on the main-chain node. Said in another way both production nodes need to be launched with the **same** key-pair.

Note: for more information about the side chain creation, refer to the document in the *request-side-chain section*.

7.2.1 Side chain configuration

Two configuration files must be placed in the configuration folder of the side chain, this is also the folder from which you will launch the node:

- appsettings.json
- appsettings.SideChain.MainNet.json

After the *release* of the side chain creation request, the **ChainId** of the new side chain will be accessible in the **SideChainCreatedEvent** logged by the transaction that released.

In this example, we will set up the side chain node with **tDVV** (1866392 converted to base58) as it's chain id, connecting to Redis' **db2**. The web API port is **1235**. Don't forget to change the **account**, **password** and **initial miner**.

If at the time of launching the side chain the P2P addresses of the other peers is known, they should be added to the bootnodes in the configuration of the side chain.

In **appsettings.json** change the following configuration sections:

```
{
  "ChainId":"tDVV",
  "ChainType":"SideChain",
  "NetType": "MainNet",
  "ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=2",
    "StateDb": "redis://localhost:6379?db=2"
  },
  "Account": {
    "NodeAccount": "YOUR PRODUCER ACCOUNT",
    "NodeAccountPassword": "YOUR PRODUCER PASSWORD"
  },
  "Kestrel": {
    "EndPoints": {
        "Http": {
            "Url": "http://*:1235/"
        }
    }
  },
  "Consensus": {
    "MiningInterval": 4000,
    "StartTimestamp": 0
  },
}
```

In **appsettings.SideChain.MainNet.json** change the following configuration sections:

```
{
  "CrossChain": {
    "Grpc": {
      "ParentChainServerPort": 5010,
      "ListeningPort": 5000,
      "ParentChainServerIp": "127.0.0.1"
    },
    "ParentChainId": "AELF",
  }
}
```

Change **ParentChainServerIp** and **ParentChainServerPort** depending on the listening address of your mainchain node.

7.2.2 Launch the side chain node

Open a terminal and navigate to the folder where you created the configuration for the side chain.

```
dotnet ../AElf.Launcher.dll
```

You can try out a few commands from another terminal to check if everything is fine, for example:

```
aelf-command get-blk-height -e http://127.0.0.1:1235
```

CHAPTER 8

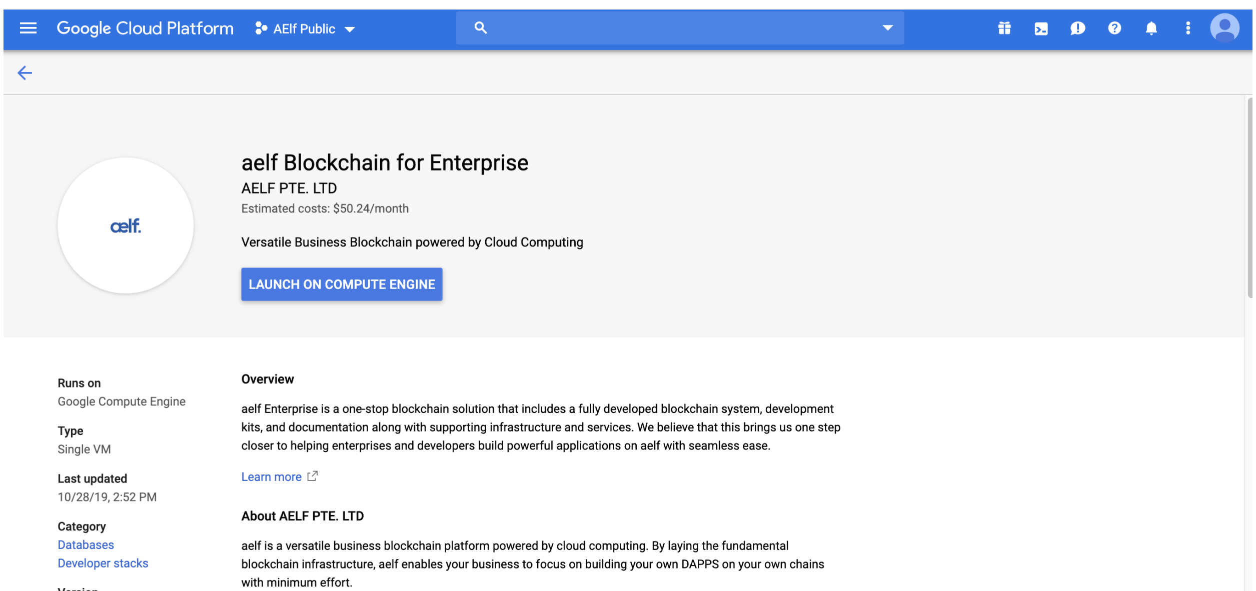
Running AElf on the cloud

This section provides resources for AElf on the cloud.

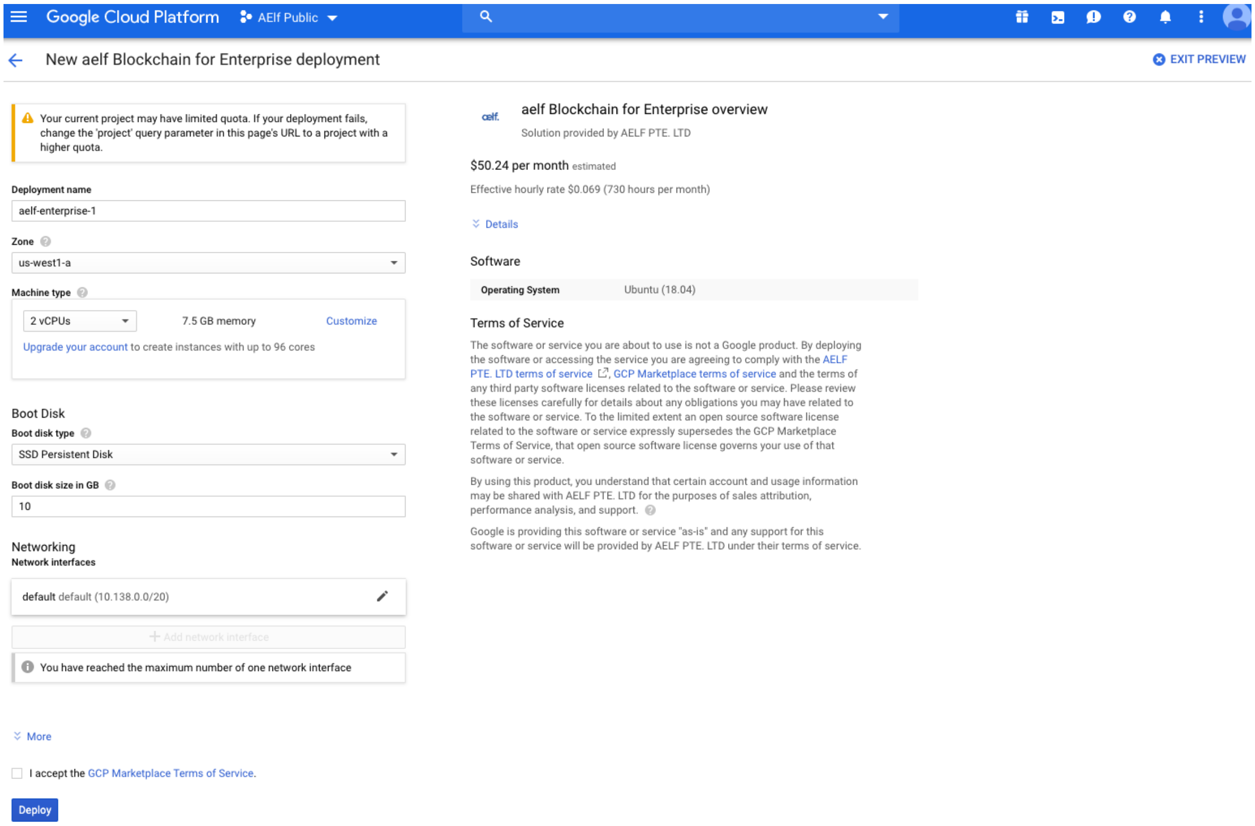
8.1 Getting started with Google cloud

This guide will run you through the steps required to run an AElf node on Google cloud (click the images for a more detailed view).

First go to the Google Cloud Market Place and search for "aelf blockchain for enterprise", find the image and select it, this will direct you to the image's page.

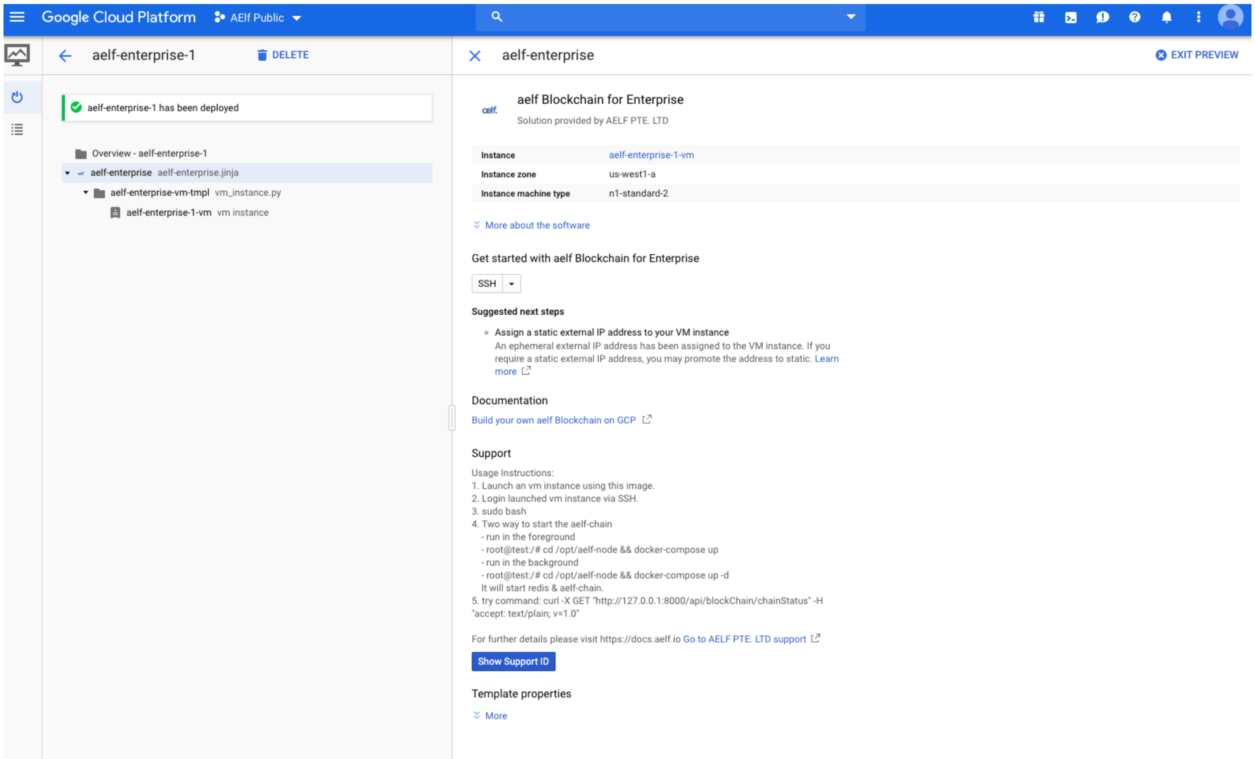


Click on the "LAUNCH ON COMPUTE ENGINE". This should bring you to the following deployment page:

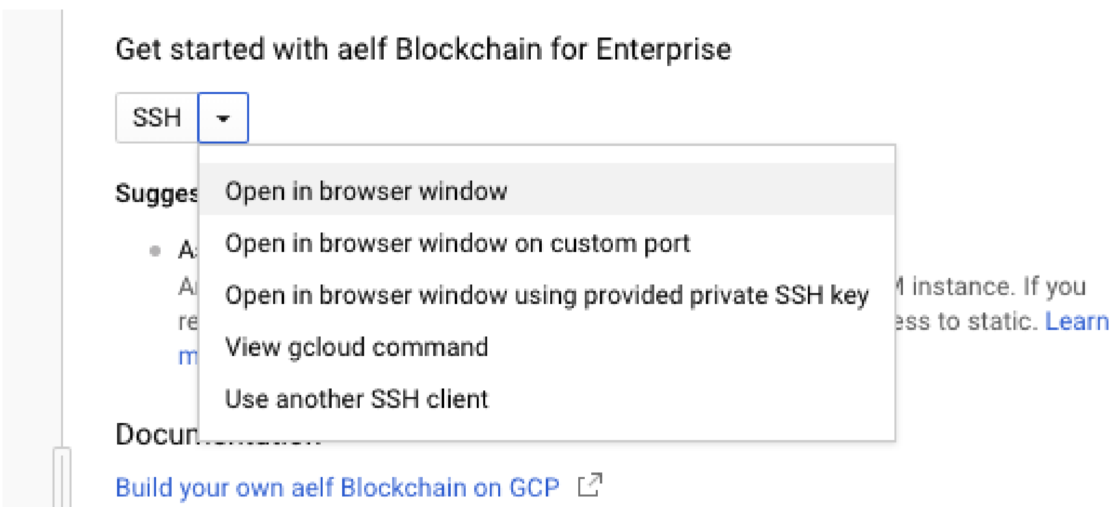


You can keep the default settings, they are sufficient to get started. If you're satisfied with the settings, just click "DEPLOY" (bottom left of the page).

This will bring you to the deployment page (wait a short moment for the instance to load), when finished you should see deployment information about the instance:



Next, login to the launched VM instance via SSH. To start the easiest way is to login to the instance directly from this deployment page. To do this click the SSH drop down and select "Open in browser window":



After loading the session, you'll get a shell to the deployed instance where you can run the chain itself.

First you'll need to execute `sudo bash` to elevate your privileges. Next, start the chain with one of the following commands (for this tutorial we'll use the second method): - either run it in the foreground: -`bash root@test:/# cd /opt/aelf-node && docker-compose up`

- or run it in the background: -`bash root@test:/# cd /opt/aelf-node && docker-compose up -d`

These commands will start redis and an AElf node (the command prints 'done' when finished).

```
ubuntu@test:/opt/aelf-node$ sudo docker-compose up -d
sudo: unable to resolve host test
Creating aelf-node_redis_1_1c73bb0fe27b ... done
Creating aelf-node_aelf-node_1_b673d69e0560 ... done
```

Finally to verify that the node is correctly working, enter the following command that will send an http request to the node in order to get the current status of the chain:

```
curl -X GET "http://127.0.0.1:8001/api/blockChain/chainStatus" -H "accept: text/plain;
↪ v=1.0"
```

```
ubuntu@test:/opt/aelf-node$ curl -X GET "http://127.0.0.1:8001/api/blockChain/chainStatus" -H "accept: text/plain; v=1.0"
{"ChainId":"AELF","Branches":{"3f41068dea72676a4de567b0098ae1bf5708d63e0d32e2745210b366a6dc0265":6727},"NotLinkedBlocks":{},"Lo
ngestChainHeight":6727,"LongestChainHash":"3f41068dea72676a4de567b0098ae1bf5708d63e0d32e2745210b366a6dc0265","GenesisBlockHash"
:"32472fa4f6a04f31f6d1c6303e7d69c496da016900559d4873a0e4c731c9f9bf","GenesisContractAddress":"2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiM
qsQ6sDG5iHT8cmjp8","LastIrreversibleBlockHash":"f89761efcf8f9f8f8c369ead32fd97ff9115bc2db5cbfaa600e7bcbc2cefa2ba","LastIrrevers
ibleBlockHeight":6703,"BestChainHash":"3f41068dea72676a4de567b0098ae1bf5708d63e0d32e2745210b366a6dc0265","BestChainHeight":6727
```

If everything is working normally you should be able to see the chain increase by repeating the last command.

CHAPTER 9

Smart Contract Developing Demos

9.1 Bingo Game

9.1.1 Requirement Analysis

Basic Requirement

Only one ruleUsers 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).

9.1.2 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

9.1.3 Write Contract

Use the code generator to generate contracts and test projects

Open the `AElf.Boilerplate.CodeGenerator` project in the *AElf.Boilerplate<https://aelf-boilerplate-docs.readthedocs.io/en/latest/usage/setup.html#try-code-generator>*, 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/dev/chain/contract/AElf.Contracts.BingoGameContract/BingoGameContract.cs

Register & Quit

Register

- 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;

- Record the user's registration time.

QuitJust delete the user's information.

Play & Bingo

Play

- Use TransferFrom to deduct the user's bet amount;
- 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);
- AmountRecord the amount of the bet 3.Record the height of the block in which the Play transaction is packaged.

Bingo

- **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;
- **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.
- **Whether the number is divisible by 2 determines the user wins or** loses;
- **The range of this number determines the amount of win/loss for the** user, see the note of GetKind method for details.

9.1.4 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);
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);
```

CHAPTER 10

Consensus

10.1 Overview

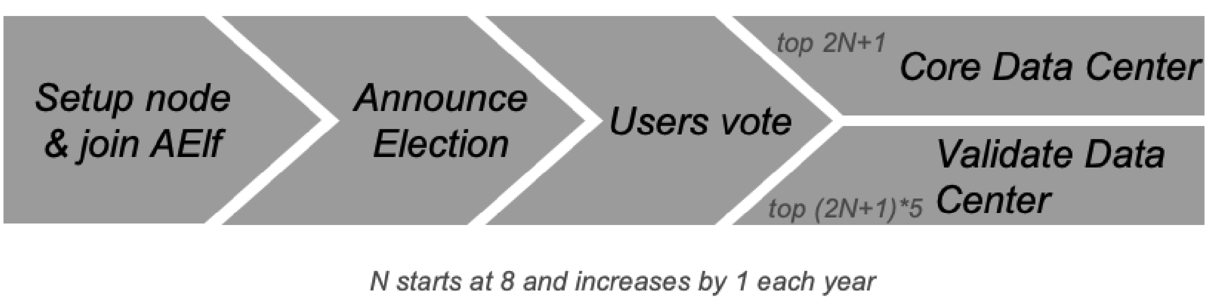
The process of reaching consensus is an essential part of every blockchain, since its what determines which transactions get included in the block and in what order. A stable and efficient Block formation mechanism is the foundation of the AElf system. The operation and maintenance of AElf is more complicated than Bitcoin and Ethereum, because AElf Block formation requires the Main Chain to record information from Side Chains, and AElf is designed to provide cloud-based enterprise services in a more complex structure. In addition, miners need to update information from multiple parallel Chains. The Main Chain will adopt AEDPoS consensus to ensure high frequency and predictability of Block formation, which will improve user experience.

In an AElf blockchain, consensus protocol is split into two parts: election and scheduling. Election is the process that determines **who** gets to produce and scheduling decides on the **when**.

10.1.1 Core Data Center

Core Data Centers aka Miners or Block Producers, act as members of parliament in the world of AElf blockchain.

The AElf blockchain delegates 2N+1 Core Data Centers. N starts with 8 and increases by 1 every year.



These nodes in the AElf system enforce all of consensus rules of AElf. The purpose of these delegated mining nodes is to enable transaction relay, transaction confirmation, packaging blocks and data transfer. As AElf adopts multi-Side

Chain architecture, Core Data Centers have to work as miners for some Side Chains. 2N+1 nodes will go through a randomized order calculation each week.

All the Core Data Centers are elected by the ELF token hodlers. Electors can lock their ELF tokens to vote to one Validate Data Center, thus enhance the competitiveness of certain Validate Data Center in the election process.

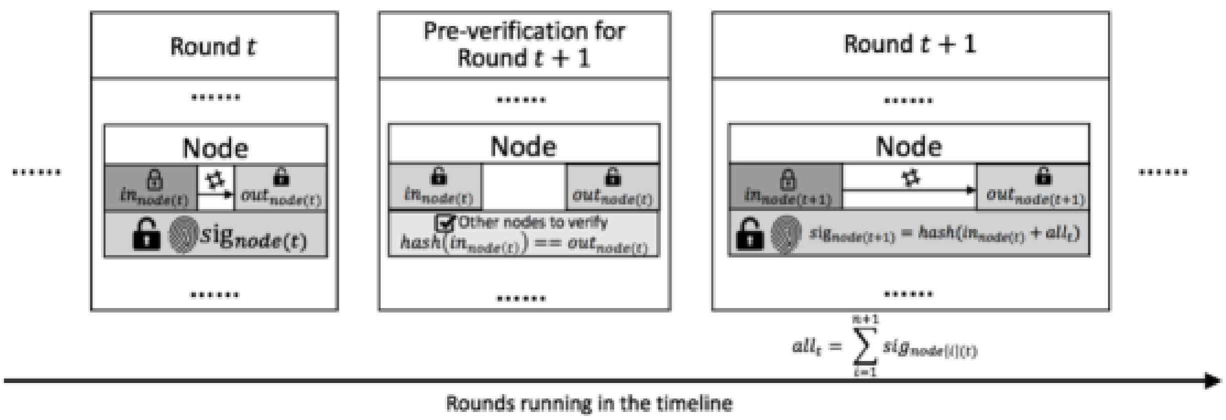
10.1.2 Validate Data Center

In the AElf blockchain, everyone can lock an amount of ELF tokens to announce himself joining the election. Among all the nodes who announced joining election, top (2N+1)*5 nodes will become Validate Data Center. N starts with 8 and increases by 1 every year.

10.2 AEDPoS Process

10.2.1 Round

The AElf blockchain is running along the timeline within processing units we call a "**round**".



In a round, one node (Core Data Center) will produce one block each time, while one node will have one extra transaction at the end of the round.

Each mining node has three main properties in a specific round **t**:

- Private key, **in_node(t)**, which is a value inputted from the mining node and kept privately by the mining node itself in round **t**. It will become public after all block generations in round **t** are completed;
- Public key, **out_node(t)**, which is the hash value of **in_node(t)**. Every node in the aelf network can look up this value at any time;
- Signature, **sig_node(t)**, which is a value generated by the mining node itself in the first round. After the first round, it can only be calculated once the previous round is completed. It is used as the signature of this mining node in this round and it is also opened to public at all times like the **out_node(t)**.

10.2.2 Main Processes

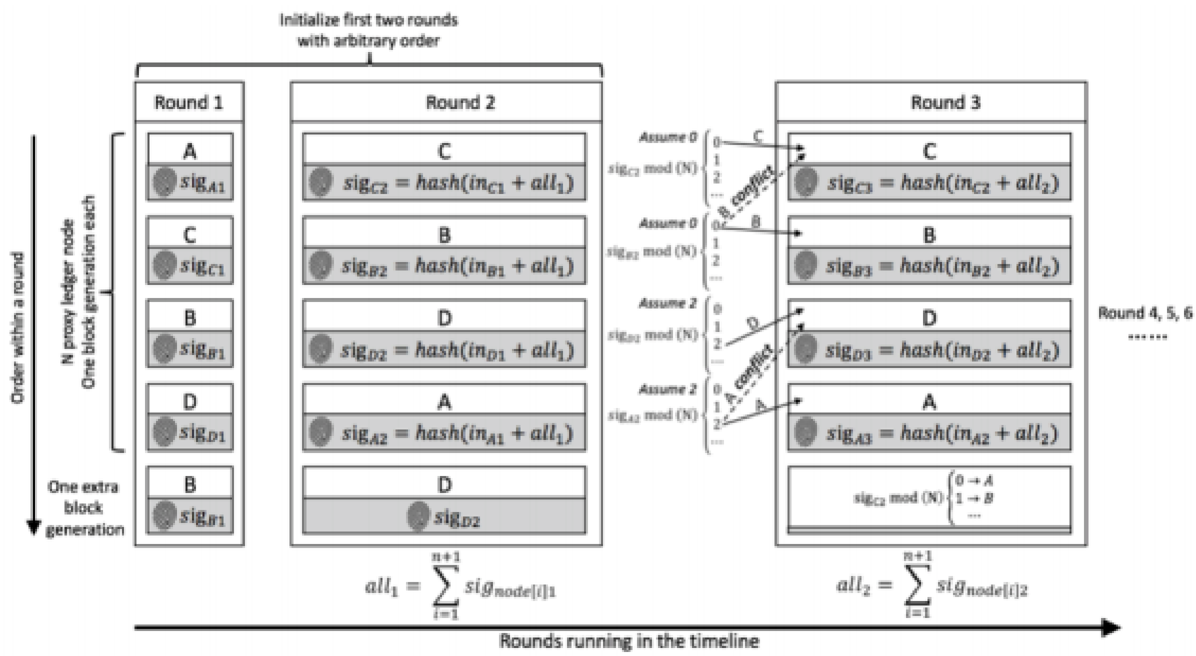
Pre-Verification

Before a node starts its block generation in round **(t+1)**, it has to have its status verified in round **t**. In round **(t+1)**, **in_node(t)** is already published as public, and **out_node(t)** can be queried at any time. So to verify the status of in round , other nodes can check **hash(in_node(t)) = out_node(t)**.

Order Calculation

In each round **N**, Core Data Centers have **(N+1)** block generation time slots, each time slot have 1 to 8 blocks generation based on current running status in the AElf blockchain.

In the first round, the ordering of block generations as well as the signature (**sig**) for each node are totally arbitrary.



In the second round, the block generations are again arbitrarily ordered. However, from the second round, the signature will be calculated by **sig_node(t+1) = hash(in_node(t) + all_t)** where

$$all_t = \sum_{i=1}^{n+1} sig_{node[i](t)}$$

here **node[i][t]**, means the node is processing the **i-th** transaction in round **t**.

From round 3, the ordering within a round is generated from the ordering and the node signature from the previous round.

In round **(t+1)**, we traverse the signature of nodes at round **t** in order. The ordering of a node in **(t+1)** is calculated by

$$sig_{node(t)} mod(N) = \begin{cases} 0, & first\ place \\ 1, & second\ place \\ 2, & third\ place \\ \cdots & \\ n-1, & n^{th}\ place \end{cases}$$

For cases of conflict, i.e. results pointed to places which are not empty, we point the node to the next available place. If the node conflict is at the **n-th** place, we will find the available place from the first place.

The node that processes the one extra transaction is calculated from the signature of the node in first place of the previous round.

$$sig_{node[0](t)} mod(N) = \begin{cases} 0, & A \\ 1, & B \\ 2, & C \\ \cdots & \end{cases}$$

sig_node[0][t] is decided by:

- all the signatures from previous round **(t-1)**;
- the **in** value of itself in round **(t-1)**;
- which node generate the extra block.

So it can only be calculated after the previous round **(t-1)** completed. Moreover, as it needs all the signatures from the previous round and the **in** value is input by each node independently, there is no way to control the ordering. The extra block generation is used to increase the randomness. In general, we create a random system that relies on extra inputs from outside. Based on the assumption that no node can know all other nodes' inputs in a specific round, no one node could control the ordering.

If one node cannot generate a block in round **t**, it also cannot input **in** its for this round. In such a case, the previous **in** will be used. Since all mining nodes are voted to be reliable nodes, such a situation should not happen often. Even if this situation does happen, the above-mentioned strategy is more than sufficient at dealing with it.

Every node only has a certain time T seconds to process transactions. Under the present network condition, T=4 is a reasonable time consideration, meaning that every node only has 4 seconds to process transactions and submit the result to the network. Any delegate who fails to submit within 4 seconds is considered to be abandoning the block. If a delegate failed two times consecutively, there will be a window period calculated as W hours (W=2^N, N stands for the number of failure) for that node.

In the systematic design, aelf defines that only one node generates blocks within a certain period. Therefore, it is unlikely for a fork to happen in an environment where mining nodes are working under good connectivity. If multiple orphan node groups occur due to network problems, the system will adopt the longest chain since that is 19 the chain that most likely comes from the orphan node group with largest number of mining nodes. If a vicious node mines in two forked Blockchains simultaneously to attack the network, that node would be voted out of the entire network.

AEDPoS mining nodes are elected in a way that resembles representative democracy. The elected nodes decide how to hand out bonuses to the other mining nodes and stakeholders.

10.3 Irreversible Block

Which means there're always some block links (a block height to its hash value) can never be reversible.

The block link currently is double confirmed by the AEDPoS mechanism during the Round changes.

CHAPTER 11

Network

11.1 Introduction

The role that the network layer plays in AElf is very important, it maintains active and healthy connections to other peers of the network and is of course the medium through which nodes communicate and follow the chain protocol. The network layer also implements interfaces for higher-level logic like the synchronization code and also exposes some functionality for the node operator to administer and monitor network operations.

The design goals when designing AElf's network layer was to avoid "reinventing the wheel" and keep things as simply possible, we ended up choosing gRPC to implement the connections in AElf. Also, it was important to isolate the actual implementation (the framework used) from the contract (the interfaces exposed to the higher-level layers) to make it possible to switch implementation in the future without breaking anything.

11.2 Architecture

This section will present a summary of the different layers that are involved in network interactions.

The network is split into 3 different layers/projects, namely:

- AElf.OS
  - Defines event handles related to the network.
  - Defines background workers related to the network.
- AElf.OS.Core.Network
  - Defines service layer exposed to higher levels.
  - Contains the definitions of the infrastructure layer.
  - Defines the component, types.
- AElf.OS.Network.Grpc
  - The implementation of the infrastructure layer.

– Launches events defined in the core
– Low-level functionality: serialization, buffering, retrying…

11.2.1 AElf.OS

At the AElf.OS layer, the network monitors events of interest to the network through event handlers, such as kernel layer transaction verification, block packaging, block execution success, and discovery of new libs. The handler will call NetworkService to broadcast this information to its connected peer. And it will run background workers to process network tasks regularly.

Currently, the AElf.OS layer handles those events related to the network:

- Transaction Accepted Eventthe event that the transaction pool receives the transaction and passes verification
- Block Mined Eventwhen the current node is BP, the event that the block packaging is completed.
- Block Accepted Eventthe event that the node successfully executes the block.
- New Irreversible Block Found Eventthe event that the chain found the new irreversible block.

Currently, the AElf.OS layer will periodically process the following tasks.

- Peer health check: regularly check whether the connected peer is healthy and remove the abnormally connected peer.
- Peer retry connection: peer with abnormal connection will try to reconnect.
- Network node discovery: regularly discover more available nodes through the network.

11.2.2 AElf.OS.Core.Network

AElf.OS.Core.Network is the core module of the networkcontains services(service layer exposed to higher levels (OS)) and definitions (abstraction of the Infrastructure layer).

- Application layer implementation:
  - NetworkService: this service exposes and implements functionality that is used by higher layers like the sync and RPC modules. It takes care of the following:
    - sending/receiving: it implements the functionality to request a block(s) or broadcast items to peers by using an IPeerPool to select peers. This pool contains references to all the peers that are currently connected.
    - handling network exceptions: the lower-level library that implements the Network layer is expected to throw a NetworkException when something went wrong during a request.
- Infrastructure layer implementation and definition:
  - IPeerPool/PeerPool: manages active connections to peers.
  - IPeer: an active connection to a peer. The interface defines the obvious request/response methods, it exposes a method for the NetworkService to try and wait for recovery after some network failure. It contains a method for getting metrics associated with the peer. You can also access information about the peer itself (ready for requesting, IP, etc.).
  - IAElfNetworkServer: manages the lifecycle of the network layer, implements listening for connections, it is the component that accepts connections. For now, it is expected that this component launches NetworkInitializationFinishedEvent when the connection to the boot nodes is finished.
- Definitions of types (network_types.proto and partial).

- Defines the event that should be launched from the infrastructure layer's implementation.

11.2.3 AElf.OS.Network.Grpc

The AElf.OS.Network.Grpc layer is the network infrastructure layer that we implement using the gRPC framework.

- GrpcPeerimplemented the interface IPeer defined by the AElf.OS.Core.Network layer
- GrpcNetworkServer: implemented the interface IAElfNetworkServer defined by the AElf.OS.Core.Network layer
- GrpcServerService: implemented network service interfaces, including interfaces between nodes and data exchange.
- Extra functionality:
  - Serializing requests/deserializing responses (protobuf).
  - Some form of request/response mechanism for peers (optionally with the timeout, retry, etc).
  - Authentification.

In fact, gRPC is not the only option. Someone could if they wanted to replace the gRPC stack with a low-level socket API (like the one provided by the dotnet framework) and re-implement the needed functionality. As long as the contract (the interface) is respected, any suitable framework can be used if needed.

11.3 Protocol

Each node implements the network interface protocol defined by AElf to ensure normal operation and data synchronization between nodes.

11.3.1 Connection

DoHandshake

When a node wants to connect with the current node, the current node receives the handshake information of the target node through the interface DoHandshake. After the current node verifies the handshake information, it returns the verification result and the handshake information of the current node to the target node.

The handshake information, in addition to being used in the verification of the connection process, will also record the status of the other party's chain after the connection is successful, such as the current height, Lib height, etc.

```
rpc DoHandshake (HandshakeRequest) returns (HandshakeReply) {}
```

- Handshake Message

  ```
  message Handshake {
      HandshakeData handshake_data = 1;
      bytes signature = 2;
      bytes session_id = 3;
  }
  ```

  - handshake_data: the data of handshake.
  - signature: the signatrue of handshake data.
  - session_id: randomly generated ids when nodes connect.

- HandshakeData Message

```
message HandshakeData {
    int32 chain_id = 1;
    int32 version = 2;
    int32 listening_port = 3;
    bytes pubkey = 4;
    aelf.Hash best_chain_hash = 5;
    int64 best_chain_height = 6;
    aelf.Hash last_irreversible_block_hash = 7;
    int64 last_irreversible_block_height = 8;
    google.protobuf.Timestamp time = 9;
}
```

  - chain_id: the id of current chain.
  - version: current version of the network.
  - listening_port: the port number at which the current node network is listening.
  - pubkey: the public key of the current node used by the receiver to verify the data signature.
  - best_chain_hash: the lastest block hash of the best branch.
  - best_chain_height: the lastest block height of the best branch.
  - last_irreversible_block_hash: the hash of the last irreversible block.
  - last_irreversible_block_height: the height of the last irreversible block.
  - time: the time of handshake.

- HandshakeRequest Message

```
message HandshakeRequest {
    Handshake handshake = 1;
}
```

  - handshake: complete handshake information, including handshake data and signature.

- HandshakeReply Message

```
message HandshakeReply {
    Handshake handshake = 1;
    HandshakeError error = 2;
}
```

  - handshake: complete handshake information, including handshake data and signature.
  - error: handshake error enum.

- HandshakeError Enum

```
enum HandshakeError {
    HANDSHAKE_OK = 0;
    CHAIN_MISMATCH = 1;
    PROTOCOL_MISMATCH = 2;
    WRONG_SIGNATURE = 3;
    REPEATED_CONNECTION = 4;
    CONNECTION_REFUSED = 5;
    INVALID_CONNECTION = 6;
    SIGNATURE_TIMEOUT = 7;
}
```

- HANDSHAKE_OK: indicate no error actually; the default value.
- CHAIN_MISMATCH: the chain ID does not match.
- PROTOCOL_MISMATCH: the network version does not match.
- WRONG_SIGNATURE: the signature cannot be verified.
- REPEATED_CONNECTION: multiple connection requests were sent by the same peer.
- CONNECTION_REFUSED: peer actively rejects the connection, either because the other party's connection pool is slow or because you have been added to the other party's blacklist.
- INVALID_CONNECTION: connection error, possibly due to network instability, causing the request to fail during the connection.
- SIGNATURE_TIMEOUT: the signature data has timed out.

3.1.2 ConfirmHandshake

When the target node verifies that it has passed the current node's handshake message, it sends the handshake confirmation message again.

```
rpc ConfirmHandshake (ConfirmHandshakeRequest) returns (VoidReply) {}
```

```
message ConfirmHandshakeRequest {
}
```

11.3.2 Broadcasting

BlockBroadcastStream

The interface BlockCastStream is used to receive information about the block and its complete transaction after the BP node has packaged the block.

```
rpc BlockBroadcastStream (stream BlockWithTransactions) returns (VoidReply) {}
```

```
message BlockWithTransactions {
   aelf.BlockHeader header = 1;
   repeated aelf.Transaction transactions = 2;
}
```

- header:
- transactions:

TransactionBroadcastStream

TransactionBroadcastStream used to receive other nodes forward transaction information.

```
rpc TransactionBroadcastStream (stream aelf.Transaction) returns (VoidReply) {}
```

AnnouncementBroadcastStream

Interface AnnouncementBroadcastStream used to receive other nodes perform block after block information broadcast.

```
rpc AnnouncementBroadcastStream (stream BlockAnnouncement) returns (VoidReply) {}
```

```
message BlockAnnouncement {
   aelf.Hash block_hash = 1;
   int64 block_height = 2;
}
```

- block_hash: the announced block hash.
- block_height: the announced block height.

LibAnnouncementBroadcastStream

Interface LibAnnouncementBroadcastStream used to receive other nodes Lib changed Lib latest information broadcast.

```
rpc LibAnnouncementBroadcastStream (stream LibAnnouncement) returns (VoidReply) {}
```

```
message LibAnnouncement{
   aelf.Hash lib_hash = 1;
   int64 lib_height = 2;
}
```

- lib_hash: the announced last irreversible block hash.
- lib_height: the announced last irreversible block height.

11.3.3 Block Request

RequestBlock

The interface RequestBlock requests a single block in response to other nodes. Normally, the node receives block information packaged and broadcast by BP. However, if the block is not received for some other reason. The node may also receive BlockAnnouncement messages that are broadcast after the block has been executed by other nodes, so that the complete block information can be obtained by calling the RequestBlock interface of other peers.

```
rpc RequestBlock (BlockRequest) returns (BlockReply) {}
```

- BlockRequest Message

  ```
  message BlockRequest {
      aelf.Hash hash = 1;
  }
  ```

  - hash: the block hash that you want to request.

- BlockReply Message

```
message BlockReply {
    string error = 1;
    BlockWithTransactions block = 2;
}
```

- error: error message.
- block: the requested block, including complete block and transactions information.

RequestBlocks

The interface RequestBlock requests blocks in bulk in response to other nodes. When a node forks or falls behind, the node synchronizes blocks by bulk fetching a specified number of blocks to the RequestBlocks interface through which the target node is called.

```
rpc RequestBlocks (BlocksRequest) returns (BlockList) {}
```

- BlocksRequest Message

```
message BlocksRequest {
    aelf.Hash previous_block_hash = 1;
    int32 count = 2;
}
```

  - previous_block_hash: the previous block hash of the request blocks, and the result does not contain this block.
  - count: the number of blocks you want to request.

- BlockList Message

```
message BlockList {
    repeated BlockWithTransactions blocks = 1;
}
```

  - blocks: the requested blocks, including complete blocks and transactions information.

11.3.4 Peer Management

Ping

Interface Ping is used between nodes to verify that each other's network is available.

```
rpc Ping (PingRequest) returns (PongReply) {}
```

```
message PingRequest {
}
```

```
message PongReply {
}
```

CheckHealth

The interface CheckHealth is invoked for other nodes' health checks, and each node periodically traverses the available peers in its own Peer Pool to send health check requests and retries or disconnects if an exception in the Peer state is found.

```
rpc CheckHealth (HealthCheckRequest) returns (HealthCheckReply) {}
```

```
message HealthCheckRequest {
}
```

```
message HealthCheckReply {
}
```

CHAPTER 12

Address

12.1 Overview

The changes of the state of an AElf blockchain are driven by the execution of transactions. An Address can identify one of the participants of a transaction, that is, either transaction sender or destination. The sender is marked as From in a transaction, and the destination is marked as To.

Actually, From can be a User Address, a Contract Address, or a Virtual Address, but To can only be a Contract Address, which means the transaction sender wants to construct a transaction to execute a certain method in that Smart Contract.

Here are some further explanations of all kinds of Address in an AElf blockchain.

12.2 User Address

User Address is generated from one key pair instance. One key pair is possessed by a real user of this AElf blockchain.

This is the defination of interface `IAElfAsymmetricCipherKeyPair`.

```
public interface IAElfAsymmetricCipherKeyPair
{
    byte[] PrivateKey { get; }
    byte[] PublicKey { get; }
}
```

Currently, in AElf blockchain, we use ECKeyPair to implement this interface, just like most of other blockchain systems. Users can use *aelf-command* tool to generate themselves a valid ECKeyPair, thus generate a unique User Address.

User can easily create a key pair with **command line tool** with the **create** command.

```
aelf-command create
```

Creation will be successful after you provide a valid password. When creating the key-pair (that we sometimes refer to as the "account") it will generate a file with the ".json" extension. This file will contain the public and private key and will be encrypted with the password you provided before.

If you are writing a dApp you can also use the following method in the *js-sdk*', it is based on bip39 for generating a deterministic key pair with a "mnemonic sentence" :

```
import Aelf from 'aelf-sdk';
Aelf.wallet.createNewWallet();
```

This will return an object containing the mnemonic used, the key-pair and the address. In AElf we usually encode the address in base58. This address is derived from the public, we calculate it as the first 30 bytes of the double sha256 hash. The AElf js-sdk provides the following, that returns the address:

```
import Aelf from 'aelf-sdk';
const address = aelf.wallet.getAddressFromPubKey(pubKey);
```

Finally here is the Protobuf message we use for representing an address, it is often used by other types to represent addresses:

```
option csharp_namespace = "AElf.Types";
message Address
{
  bytes value = 1;
}
```

Also, the structure of Hash is very similar to Address.

12.3 Contract Address

Contract Address can identify a Smart Contract in an AElf blockchain. The Contract Address is calculated with chain id and a serial number during the deployment of related Smart Contract.

```
private static Address BuildContractAddress(Hash chainId, long serialNumber)
{
    var hash = HashHelper.ConcatAndCompute(chainId, HashHelper.
↪ComputeFrom(serialNumber));
    return Address.FromBytes(hash.ToByteArray());
}
public static Address BuildContractAddress(int chainId, long serialNumber)
{
    return BuildContractAddress(HashHelper.ComputeFrom(chainId), serialNumber);
}
```

12.4 Contract Virtual Address

As an extended function, every contract can be added with a Hash value based on its Address, then it can obtain unlimited virtual Addresses, this newly created address is called **Virtual Address**.

For example, the account transfer in AEif blockchain is to send the **Transfer** transaction to the MultiToken contract along with the parameters of the recipient, transfer currency and amount, etc. One account transfer involves the sender and recipient, and both parties are identified by the Address. In this situation, the Virtual Address, which is created by Address and Hash algorithm, can be either party of the account transfer like the normal Address for the user or contract. What's more, Virtual Address can only be controlled by the primary contract, this enables the contract to custody transactions or fundings independently for every user.

In essence, the characteristic of Virtual Address is a unique identification. As a result, the Virtual Address, which is generated by a business action on this contract, is reliable to be used for token transferring.

CHAPTER 13

Overview

Transactions ultimately are what will change the state of the blockchain by calling methods on smart contracts. A transaction is either sent to the node via RPC or received from the network. When broadcasting a transaction and if valid it will be eventually included in a block. When this block is received and executed by the node, it will potential change the state of contracts.

13.1 Smart Contract

In AElf blockchain, smart contracts contains a set of **state** definitions and a set of methods which aiming at modifing these **state**s.

13.2 Action & View

In AElf blockchain, there are two types of smart contract methods, actions and views. Action methods will actually modify the state of one contract if a related transaction has included in a block and executed successfully. View methods cannot modify the state of this contract in any case.

Developers can claim a action method in proto file like this:

```
rpc Vote (VoteInput) returns (google.protobuf.Empty) {
}
```

And claim a view method like this:

```
rpc GetVotingResult (GetVotingResultInput) returns (VotingResult) {
    option (aelf.is_view) = true;
}
```

13.3 Transaction Instance

Here's the defination of the Transaction.

```
option csharp_namespace = "AElf.Types";

message Transaction {
    Address from = 1;
    Address to = 2;
    int64 ref_block_number = 3;
    bytes ref_block_prefix = 4;
    string method_name = 5;
    bytes params = 6;
    bytes signature = 10000;
}
```

In the js sdk, there are multiple methods to work with transactions. One important method is the **getTransaction** method that will build a transaction object for you:

```
import Aelf from 'aelf-sdk';
var rawTxn = proto.getTransaction('65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9'
↪'65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9', 'SomeMethod', encodedParams);
```

This will build the transaction to the contract at address "65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9" that will call **SomeMethod** with encoded params.

13.3.1 From

The address of the sender of a transaction.

Note that the **From** is not currently useful because we derive it from the signature.

13.3.2 To

The address of the contract when calling a contract.

13.3.3 MethodName

The name of a method in the smart contract at the **To** address.

13.3.4 Params

The parameters to pass to the aforementioned method.

13.3.5 Signature

When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.

You can use the js-sdk to sign the transaction with the following method:

```
import Aelf from 'aelf-sdk';
var txn = Aelf.wallet.signTransaction(rawTxn, wallet.keyPair);
```

13.3.6 RefBlockNumber & RefBlockPrefix

These two fields measure whether this transaction has expired. The transaction will be discarded if it is too old.

13.4 Transaction Id

The unique identity of a transaction. Transaction Id consists of a cryptographic hash of the instance basic fields, excluding signature.

Note that the Transaction Id of transactions will be the same if the sender broadcasted several transactions with the same origin data, and then these transactions will be regarded as one transaction even though broadcasting several times.

13.4.1 Verify

One transaction now is verified by the node before forwarding this transaction to other nodes. If the transaction execution is failed, the node won't forward this transaction nor package this transaction to the producing block.

We have several transaction validationi providers such as:

- BasicTransactionValidationProvider. To verify the transaction signature and size.
- TransactionExecutionValidationProvider. To pre-execute this transaction before forwarding this transaction or really packaging this transaction to new block.
- TransactionMethodValidationProvider. To prevent transaction which call view-only contract method from packaging to new block.

13.4.2 Execution

In AElf, the transaction is executed via .net reflection mechanism.

Besides, we have some transaction execution plugins in AElf main net. The execution plugins contain pre-execution plugins and post-execution plugins.

- FeeChargePreExecutionPlugin. This plugin is for charging method fees from transaction sender.
- MethodCallingThresholdPreExecutionPlugin. This plugin is for checking the calling threshold of a specific contract or contract method.
- ResourceConsumptionPostExecutionPlugin. This plugin is for charging resource tokens from called contract after transaction execution (thus we can know how much resource tokens are cost during the execution.)

13.4.3 TransactionResult

Data structure of TransactionResult:

```
message TransactionResourceInfo {
    repeated aelf.ScopedStatePath write_paths = 1;
    repeated aelf.ScopedStatePath read_paths = 2;
    ParallelType parallel_type = 3;
    aelf.Hash transaction_id = 4;
    aelf.Hash contract_hash = 5;
    bool is_nonparallel_contract_code = 6;
}
```

CHAPTER 14

Core

14.1 Application pattern

We follow generally accepted good practices when it comes to programming, especially those practices that make sense to our project. Some practices are related to C# and others are more general to OOP principles (like SOLID, DRY…).

Even though it's unusual for blockchain projects, we follow a domain driven design (DDD) approach to our development style. Part of the reason for this is that one of our main frameworks follows this approach and since the framework is a good fit for our needs, it's natural that we take the same design philosophy.

A few key points concerning DDD:

- traditionally, four layers: presentation, application, domain and infrastructure.
- presentation for us corresponds to any type of dApp.
- application represents exposed services mapped to the different domains.
- domain represents the specific events related to our blockchain system and also domain objects.
- finally infra are the third party libraries we use for database, networking…

We also have a Github issue where we list some of the coding standards that we follow while developing AElf.

14.1.1 Frameworks and libraries:

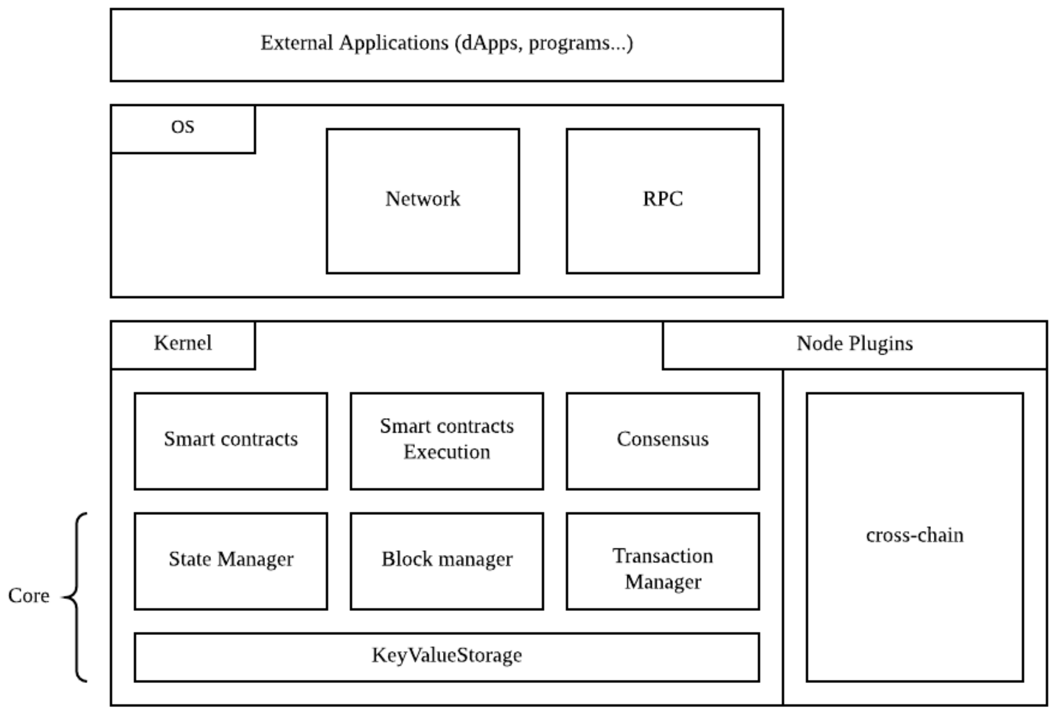
The main programming language used to code and build AElf is C# and is built with the dotnet core framework. It's a choice that was made due to the excellent performances observed with the framework. Dotnet core also comes with the benefit of being cross platform, at least for the three main ones that are Windows, MacOS and Linux. Dotnet core also is a dynamic and open source framework and comes with many advantages of current modern development patterns and is backed by big actors in the IT space.

At a higher level we use an application framework named ABP. From a functional point of view, a blockchain node is a set of endpoints, like RPC, P2P and cross-chain and some higher level protocol on top of this. So ABP is a natural fit for this, because it offers a framework for building these types of applications.

We use the XUnit framework for our unit tests. We also have some custom made frameworks for testing smart contracts.

For lower level, we use gRPC for the cross-chain and p2p network communication. Besides for gRPC, we also use Protobuf for serialization purposes.

14.2 Design principles:



The above diagram shows the conceptual structure of the node and the separation between OS and Kernel.

14.2.1 OS

The OS layer implements the application and infrastructure layer for the network. It also implements the high level handlers for network events and job, like for example synchronizing the chain in reaction to a block announcement. The OS layer also contains the RPC implementation for the exposed API.

Kernel

The kernel contains the smart contract and execution primitives and definitions. The kernel also defines the components necessary for accessing the blockchain's data. Various managers will use the storage layer to access the underlying database.

The kernel also defines the notion of plugins. The diagram show that the side chain modules are implemented as plugins.

Structure of the project:

To help follow AElf's structure this section will present you with an overview of the solution.

Conceptually, AElf is built on two main layers: OS and Kernel. The OS contains the high level definition for a node and the endpoints like RPC and p2p, whereas the kernel mainly contains logic and definitions for smart contracts and consensus.

AElf has a native runtime for smart contracts which is implemented in C# and for contracts written in C#. The implementation is the AElf.Runtime.CSharp.* projects.

A big part of AElf is the side chain framework. It is mainly implemented in the AElf.CrossChain namespace and defines the main abstractions in the **core** project and an implementation with grpc in the AElf.Crosschain.Grpc project.

The AElf.Test solution folder contains all the tests, coverage of the main functional aspects must be at a maximum to ensure the quality of our system.

Finally there are other projects that implement either libraries we use, like the crypto library and others for infrastructure like the database library, that are not as important but are still worth looking into.

14.2.2 Jobs and event handlers

Event handlers implement the logic that reacts to external in internal events. They are in a certain sense the higher levels of the application (they are called by the framework in purely domain agnostic way). An event handler, mostly using other services will influence the state of the chain.

14.2.3 Modules

We currently base our architecture on modules that get wired together at runtime. Any new module must inherit **AElfModule**.

Give the need to implement a new module, it usually follows the following steps: 1. Write the event handler or the job. 2. implement the interface and create manager or infrastructure layer interface that is needed. 3. implement the infrastructure layer interface in the same project in it do not need add dependency. 4. implement the infrastructure layer interface in another project, if it need third party dependency, for example, you can add GRPC / MongoDB / MySQL in the new project.

Example: the p2p network module.

The networking code is defined amongst 2 modules: **CoreOSAElfModule** and **GrpcNetworkModule**. The OS core defines the application service (used by other components of the node) and also implements it since it is application/domain logic. Whereas the infrastructure layer (like the server endpoint), is defined in the OS core modules but is implemented in another project that relies on a third party - gRPC in this case.

14.2.4 Testing

When writing a new component, event handler, method. . . It's important for AElf's quality to consider the corresponding unit test. As said previously we have a solution-wide test folder where we place all the tests.

CHAPTER 15

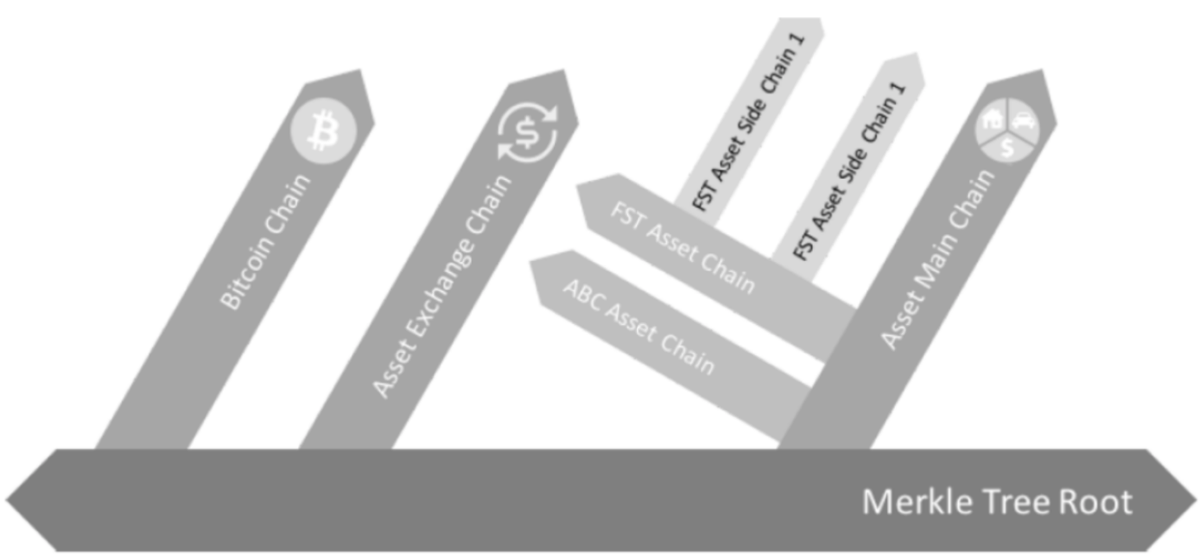
Cross Chain

15.1 Introduction

One of the major issues with current blockchain systems is scalability. Mainly because of **congestion problems** of current blockchains, the problem is that when a single chain needs to sequentially order and process transactions, in the event of a popular dApp taking up a lot of resources, it has negative side effects on other dApps.

This is why AElf side chains were introduced in the initial design. It's envisioned that one side-chain is responsible for handling one or more similar business scenarios, distributing different tasks on multiple chains and improving the overall processing efficiency.

The main idea is that the side-chains are **independent** and **specialized** to ensure that the dapps running on them can perform efficiently and smoothly. A network link will exist between main-chain node and side-chain nodes, but the communication is indirectly done through what's called a Merkle root.



The diagram above illustrates the conceptual idea behind side chains.

Side chains are isolated but still need a way to interact with each other for this AElf introduces a communication mechanism through **merkle roots** and **indexing** to enable cross chain verification scenarios.

The following sections of this documentation will give you an overview of the architecture of AElf's side chains. There will also be a guide explaining how to set up a main-chain and a side chain node.

15.2 Overview

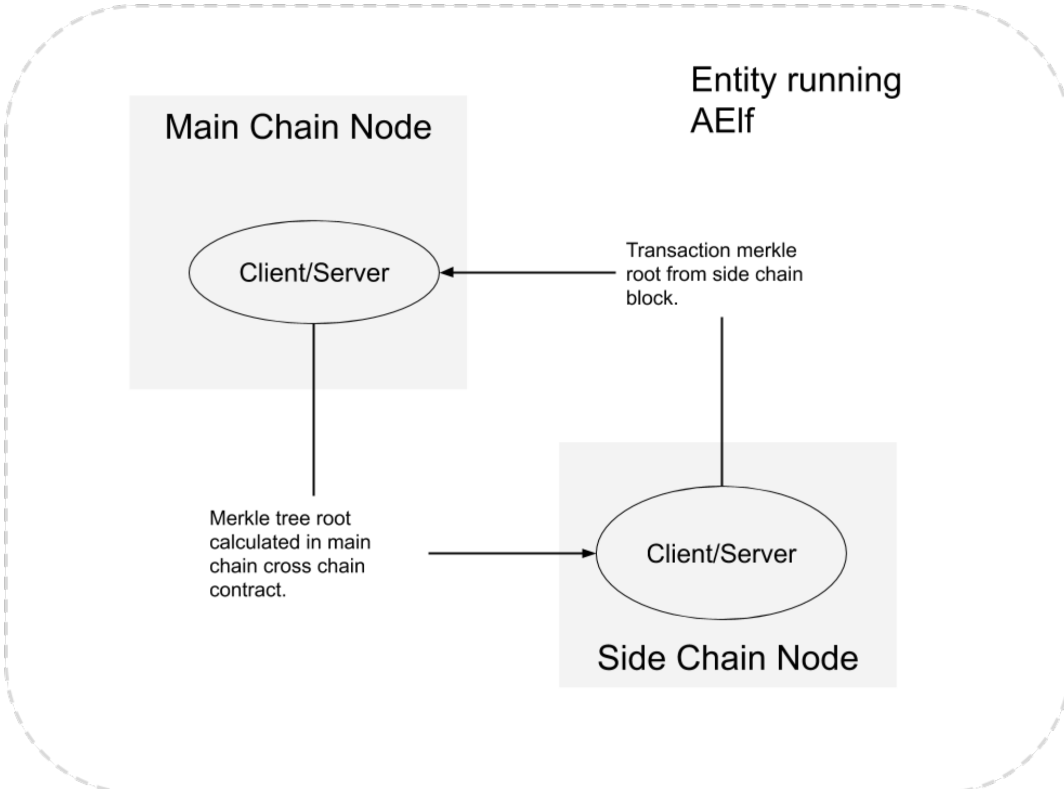
Conceptually a side chain node and main chain node are similar, they are both independent blockchains, with their own peer-to-peer network and possibly their own ecosystem. It is even possible to have this setup on multiple levels. In terms of peer-to-peer networks, all side chains work in parallel to each other but they are linked to a main chain node through a cross-chain communication mechanism.

Through this link, messages are exchanged and indexing is performed to ensure that transactions from the main-chain or other side chains are verifiable in the side chain. Implementers can use AElf libraries and frameworks to build chains.

One important aspect is the key role that the main chain plays, because its main purpose is to index the side chains. Only the main chain indexes data about all the side chains. Side chains are independent and do not have knowledge about each other. This means that when they need to verify what happened in other chains, they need the main chain as a bridge to provide the cross chain verification information.

15.2.1 Node level architecture

In the current architecture, both the side chain node and the main chain node has one server and exactly one client. This is the base for AElf's two-way communication between main chain and side chains. Both the server and the client are implemented as a node plugins (a node has a collection of plugins). Interaction (listening and requesting) can start when both the nodes have started.



The diagram above illustrates two nodes run by an entity: one main chain node and one side chain node. Note that the nodes don't have to be in the same physical location.

Side chain lifetime

Side chain lifetime involves the following steps.

- Request side chain creation.
- Wait for accept on main chain.
- Start and initialize side chain and it will be indexed by main chain automatically.
- It is allowed to do cross chain verification iff side chain is indexed correctly.

Communication

When the side chain node starts it will initiate a number of different communications, here are the main points of the protocol:

- When the side chain node is started for the first time it will request the main chain node for a chain initialization context.
- After initialization the side chain is launched and will perform a handshake with main chain node to signal that it is ready to be indexed.

- During the indexing process, the information of irreversible blocks will be exchanged between side chain and main chain. The main chain will write the final result in block which is calculated with the cross chain data from all side chains. Side chain is also recording the data in contract from main chain.

AElf provides the cross chain communication implementation with grpc.

```
    rpc RequestIndexingFromParentChain (CrossChainRequest) returns (stream acs7.
↪ParentChainBlockData) {}
    rpc RequestIndexingFromSideChain (CrossChainRequest) returns (stream acs7.
↪SideChainBlockData) {}
```

Cache

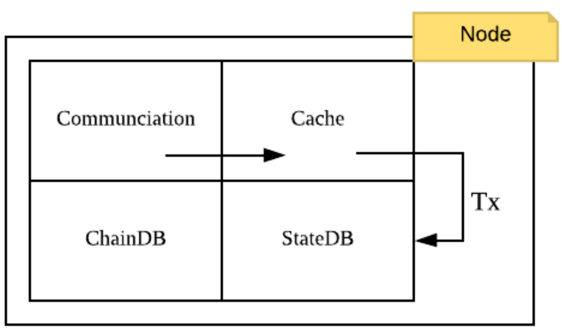
For effective indexing, a cache layer is used to store cross chain data received from remote nodes, and make it available and correct. Cross chain data is cached by chain id and block height with a count limit. The cache layer can give the data if cached when the node needs it. So cache layer decouples the communication part and node running logic.

Cross chain contract

Apart from the data in block, most cross chain data will be stored by the cross chain contract. Cross chain data cached by the node is packed in transaction during the mining process and the calculated result is stored by the contract. Actually, the cross chain data in the block is the side chain indexing result of calculations in this contract. Only with data in this contract can cross chain verification work correctly.

Data flow

Conceptually the node is like described in the following diagram. Main/Side chain node gets the cross chain data from the other side and put it in the local memory. Indexing transaction will be packed by miner and cross chain data would go into `State` through `Crosschain Contract`.



15.3 Cross chain verification

Verification is the key feature that enables side chains. Because side chains do not have direct knowledge about other side chains, they need a way to verify information from other chains. Side chains need the ability to verify that a transaction was included in another side chains block.

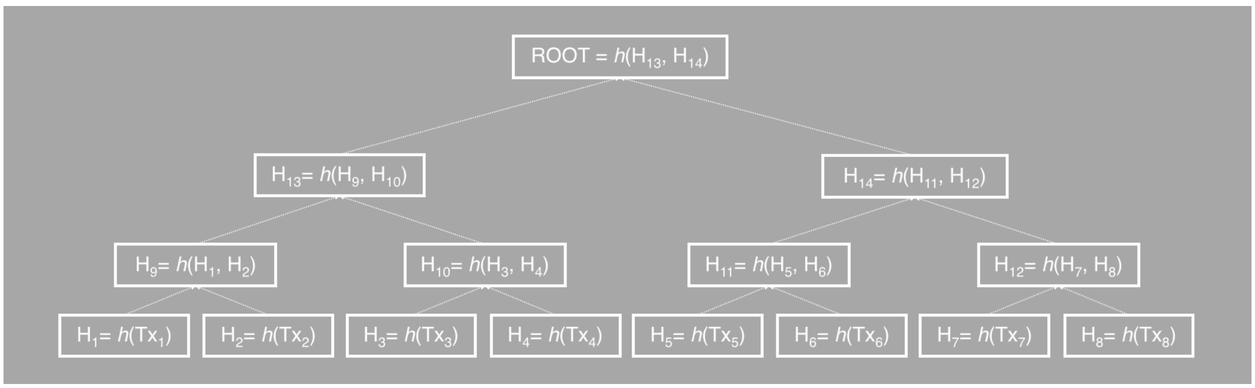
15.3.1 Indexing

The role of the main chain node is to index all the side chains blocks. This way it knows exactly the current state of all the side chains. Side chains also index main chain blocks and this is how they can gain knowledge about the inclusion of transactions in other chains.

Indexing is a continuous process, the main chain is permanently gathering information from the side chains and the side chains are permanently getting information from the main chain. When a side chain wants to verify a transaction from another side chain it must wait until the correct main chain block has been indexed.

15.3.2 Merkle tree

Merkle tree is a basic binary tree structure. For cross-chain in AElf, leaf value is the hash from transaction data. Node value (which is not a leaf node) is the hash calculated from its children values until to the tree root.

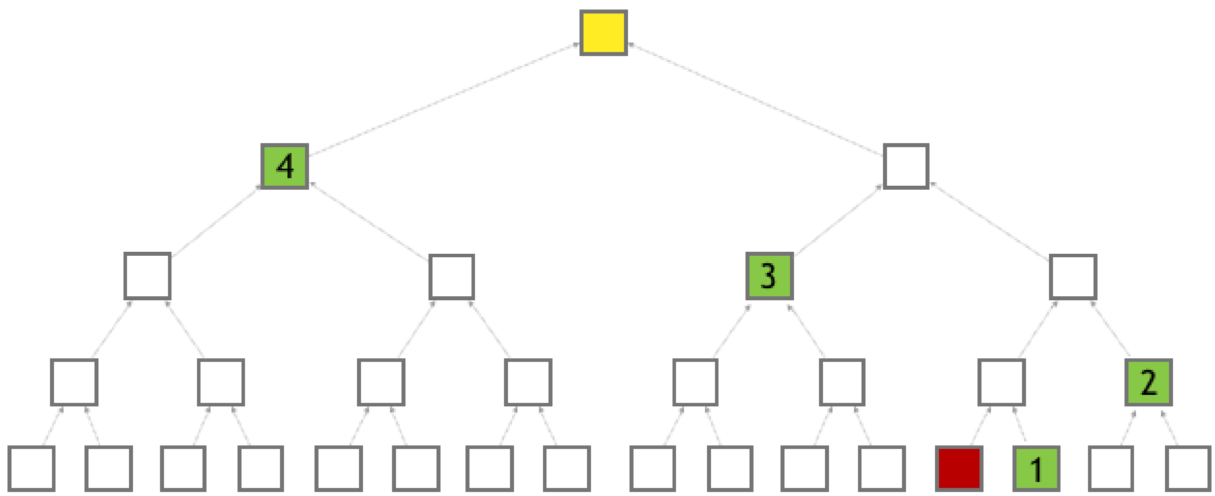


15.3.3 Merkle root

When a transaction gets included in a side chain's block the block will also include a merkle root of the transactions of this block. This root is local to this side chain's blockchain and by itself of little value to other side chains because they follow a different protocol. So communication between side chains goes through the main chain in the form of a merkle path. During indexing process, main chain is going to calculate the root with the data from side chains, and side chains in turn get the root in future indexing. This root is used for final check in cross chain transaction verification.

15.3.4 Merkle path

Merkle path is the node collection for one leaf node to calculate with to the root. Correct merkle path is necessary to complete any work related to cross chain verification. For the transaction **tx** from chain **A**, you need the whole merkle path root for **tx** to calculate the final root if you want to verify the existence of this transaction on other chains, and verify the root by checking whether it is equal to the one obtained from indexing before.



15.4 Cross chain verify

This section will explain how to verify a transaction across chains. It assumes a side chain is already deployed and been indexed by the main-chain.

15.4.1 Send a transaction

Any transaction with status `Mined` can be verified, the only pre-condition is that the transaction was indexed.

15.4.2 Verify the transaction

There's basically two scenarios that can be considered:

- verifying a main-chain transaction.
- verifying a side-chain transaction.

```
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;
}
```

VerifyTransaction is the view method of the cross-chain contract and that will be used to perform the verification. It returns whether the transaction was mined and indexed by the destination chain. This method will be used in both scenarios, what differs is the input:

Verify a main-chain tx

Verifying a main-chain transaction on a side chain, you can call **VerifyTransaction** on the side-chain with the following input values:

- parent_chain_height - the height of the block, on the main-chain, in which the transaction was packed.
- transaction_id - the ID of the transaction that you want to verify.
- path - the merkle path from the main-chain's web api with the **GetMerklePathByTransactionIdAsync** with the ID of the transaction.
- verified_chain_id - the source chainId, here the main chain's.

You can get the `MerklePath` of transaction in one block which packed it by chain's web api with the **GetMerklePathByTransactionIdAsync** (See *web api reference*).

Verify a side-chain tx

First, you also need the query result of **GetMerklePathByTransactionIdAsync**, just like verification for a main-chain tx.

And then if you want to verify a a side-chain transaction, you need to get the `CrossChainMerkleProofContext` of this tx from the source chain. You can try the **GetBoundParentChainHeightAndMerklePathByHeight** method of `Crosschain contract`.

The input of this api is the height of block which packed the transaction. And it will return merkle proof context

```
rpc GetBoundParentChainHeightAndMerklePathByHeight (google.protobuf.Int64Value)␣
↪returns (CrossChainMerkleProofContext) {
    option (aelf.is_view) = true;
}

message CrossChainMerkleProofContext {
    int64 bound_parent_chain_height = 1;
    aelf.MerklePath merkle_path_from_parent_chain = 2;
}
```

With the result returned by above api, you can call **VerifyTransaction** on the target chain with the following input values:

- transaction_id - the ID of the transaction that you want to verify.
- parent_chain_height - use the **bound_parent_chain_height** field of **CrossChainMerkleProofContext** .
- path - the concatenation of 2 merkle paths, in order:
  - the merkle path of the transaction, use the web api method **GetMerklePathByTransactionIdAsync**.
  - use the **merkle_path_from_parent_chain** field from the **CrossChainMerkleProofContext** object.
- verified_chain_id - the source chainId, here the side chain on which the transaction was mined.

15.5 Cross chain transfer

Cross chain transfer is one of mostly used cases when it comes to cross chain verification. AElf already supports cross chain transfer functionality in contract. This section will explain how to transfer tokens across chains. It assumes a side chain is already deployed and been indexed by the main chain.

The transfer will always use the same contract methods and the following two steps: - initiate the transfer - receive the tokens

15.5.1 Prepare

Few preparing steps are required before cross chain transfer, which is to be done only once for one chain. Just ignore this preparing part if already completed.

Let's say that you want to transfer token FOO from chain *A* to chain *B*. Note that please make sure you are already clear about how cross chain transaction verification works before you start. Any input contains`MerklePath` in the following steps means the cross chain verification processing is needed. See *cross chain verification* for more details.

- Validate `Token Contract` address on chain *A*.

  Send transaction *tx_1* to `Genesis Contract` with method ValidateSystemContractAddress. You should provide **system_contract_hash_name** and address of `Token Contract` . *tx_1* would be packed in block successfully.

```
rpc ValidateSystemContractAddress(ValidateSystemContractAddressInput) returns␣
↪(google.protobuf.Empty){}

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

- Register token contract address of chain *A* on chain *B*.

  Create a proposal, which is proposed to `RegisterCrossChainTokenContractAddress`, for the default parliament organization (check *Parliament contract* for more details) on chain *B*. Apart from cross chain verification context, you should also provide the origin data of *tx_1* and `Token Contract` address on chain *A*.

```
rpc RegisterCrossChainTokenContractAddress␣
↪(RegisterCrossChainTokenContractAddressInput) returns (google.protobuf.Empty) {}

message RegisterCrossChainTokenContractAddressInput{
    int32 from_chain_id = 1;
    int64 parent_chain_height = 2;
    bytes transaction_bytes = 3;
    aelf.MerklePath merkle_path = 4;
    aelf.Address token_contract_address = 5;
}
```

- Validate `TokenInfo` of *FOO* on chain *A*.

  Send transaction *tx_2* to `Token Contract` with method `ValidateTokenInfoExists` on chain *A*. You should provide `TokenInfo` of *FOO*. *tx_2* would be packed in block successfully.

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

message ValidateTokenInfoExistsInput{
    string symbol = 1;
    string token_name = 2;
    int64 total_supply = 3;
    int32 decimals = 4;
```

(continues on next page)

(continued from previous page)

```
    aelf.Address issuer = 5;
    bool is_burnable = 6;
    int32 issue_chain_id = 7;
}
```

- Create token *FOO* on chain *B*.

  Send transaction *tx_3* to `Token Contract` with method CrossChainCreateToken on chain *B*. You should provide the origin data of *tx_2* and cross chain verification context of *tx_2*.

```
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;
}
```

15.5.2 Initiate the transfer

On the token contract of source chain, it's the `CrossChainTransfer` method that is used to trigger the transfer:

```
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;
}
```

The fields of the input:

- **to** - the target address to receive token
- **symbol** - symbol of token to be transferred
- **amount** - amount of token to be transferred
- **memo** - memo field in this transfer
- **to_chain_id** - destination chain id on which the tokens will be received
- **issue_chain_id** - the chain on which the token was issued

15.5.3 Receive on the destination chain

On the destination chain tokens need to be received, it's the `CrossChainReceiveToken` method that is used to trigger the reception:

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

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

rpc GetBoundParentChainHeightAndMerklePathByHeight (aelf.Int64Value) returns␣
↪(CrossChainMerkleProofContext) {
    option (aelf.is_view) = true;
}

message CrossChainMerkleProofContext {
    int64 bound_parent_chain_height = 1;
    aelf.MerklePath merkle_path_from_parent_chain = 2;
}
```

Let's review the fields of the input

- **from_chain_id**

  the source chain id on which cross chain transfer launched

- **parent_chain_height**

  - for the case of transfer from main chain to side chain: this parent_chain_height is the height of the block on the main chain that contains the `CrossChainTransfer` transaction.
  - for the case of transfer from side chain to side chain or side chain to main-chain: this **parent_chain_height** is the result of **GetBoundParentChainHeightAndMerklePathByHeight** (input is the height of the *CrossChainTransfer*, see *cross chain verification*) - accessible in the **bound_parent_chain_height** field.

- **transfer_transaction_bytes**

  the serialized form of the `CrossChainTransfer` transaction.

- **merkle_path**

  You should get this from the source chain but merkle path data construction differs among cases.

  - for the case of transfer from main chain to side chain
    - only need the merkle path from the main chain's web api `GetMerklePathByTransactionIdAsync` (`CrossChainTransfer` transaction ID as input).
  - for the case of transfer from side chain to side chain or from side chain to main chain
    - the merkle path from the source chain's web api `GetMerklePathByTransactionIdAsync` (`CrossChainTransfer` transaction ID as input).
    - the output of `GetBoundParentChainHeightAndMerklePathByHeight` method in `Cross chain Contract` (`CrossChainTransfer` transaction's block height as input). The path nodes are in the **merkle_path_from_parent_chain** field of the `CrossChainMerkleProofContext` object.
    - Concat above two merkle path.

CHAPTER 16

Smart contract

16.1 Smart contract architecture

At its core, a blockchain platform can be viewed as a distributed multi-tenant database that holds the state of all the smart contracts deployed on it. After deployment, each smart contract will have a unique address. The address is used to scope the state and as the identifier for state queries and updates. The methods defined in the smart contract code provides the permission checks and logics for queries and updates.

In aelf, a smart contract essentially has three parts: the interface, the state, and the business logic.

1. **the interface** - aelf supports smart contracts coded in multiple languages. Protobuf format is adopted as the cross-language definition of the contract.
2. **the state** - the language specific SDK provides some prototypes for the state of different types, after the definition of properties of certain prototype, developers could query and update *state database* via accessing the properties directly.
3. **the business logic** - aelf provides protobuf plugins to generate the smart contract skeleton from the contract's proto definition. Developers just need to fill the logics for each method by override.

Smart contracts in AElf are spread across the Kernel, the runtime and the SDK. The kernel defines the fundamental components and infrastructure associated with smart contracts. It also defines the abstractions for execution. Smart contract also heavily rely on the runtime modules and the sdk project.

Smart contracts, along with the blockchain's data, form the heart of a blockchain system. They define through some predefined logic how and according to what rules the state of the blockchain is modified.

A smart contract is a collection of methods that each act upon a particular set of state variables.

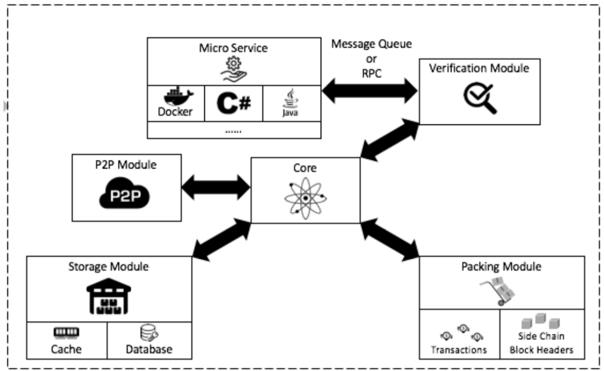
Transactions trigger the logic contained in smart contracts. If a user of the blockchain wants to modify some state, he needs to build a transaction that will call a specific method on some contract. When the transaction is included in a block and this block is executed, the modifications will be executed.

Smart contracts are a part of what makes dApps possible. They implement a part of the business layer: the part that gets included in the blockchain.

What follows in this section will give you a general overview of how AElf implements smart contracts. The other sections will walk you through different notions more specifically.

16.1.1 Architecture overview

In AElf, Smart Contracts are defined like micro-services. This makes Smart Contracts independent of specific programming languages. This implies, for example, that our Consensus Protocol essentially becomes a service because it is defined through Smart Contract.



As showed in the diagram above, smart contracts functionality is defined within the kernel. The kernel defines the fundamental components and infrastructure associated with establishing smart contracts as a service: * SDK abstracts - high-level entities that provide a hook for smart contract services to interact with the chain. * Execution - high-level primitives defined for execution

16.1.2 Chain interactions

Smart contract need to interact with the chain and have access to contextual information. For this AElf defines a bridge and a bridge host. Usually the programming SDK corresponding to the specific language will implement features to communicate with/through the bridge.

One of the major functionalities provided by the bridge is the ability to provide contextual information to the smart contract being executed. Here are a few: the **Self** field represents the address of the current contract being called. the **Sender** is the address that sent the transaction that executed the contract, and **Origin** is the address that signed the transaction. Sometimes **Sender** and **Origin** are equal.the **OriginTransactionId** is the ID of the transaction fetch from transaction pool or generated by the current miner, and **TransactionId** is the Id of the transaction is executing, which means this transaction could be an inline one.

The bridge also exposes extra functionality: contracts can fire **Events**, which are in a way similar to logging. contracts can call a method on another contract in a read-only manner. Any state change will not be persisted to the blockchain. Send inline - this actually creates a transaction to call another method. As opposed to calling the changes to the state - if any - will be persisted.

State

The main point of a smart contract is to read and/or modify state. The language SDK's implement state helpers and through the bridge's **StateProvider**.

16.1.3 Runtime and execution

When a block's transactions are executed, every transaction will generate a trace. Amongst other things, it contains: the return value of the called method, this can be anything defined in protobuf format and is defined in the service definition. error outputs, if execution encountered a problem. the results from inner calls in **InlineTraces** field. the **Logs** field will contain the events launched from the called method.

16.1.4 Sdk

AElf comes with a native C# SDK that gives smart contract developers the necessary tools to develop smart contracts in C#. It contains helpers to communicate with the bridge. By using the SDK, you can also take advantage of the type infrastructure defined in the library: ContractState: an interface that is implemented by a class that is destined to be containers for the state field. MappedState: a base type that defines **collections** a key-value mapping, generic subclasses are available to enable multi-key scenarios. SingletonState: this defines **non-collection** types with a

Any developer or company can develop an sdk and a runtime for a specific language by creating an adapter to communicate with the bridge through gRPC.

16.2 Smart contract service

When writing a smart contract in AElf the first thing that need to be done is to define it so it can then be generate by our tools. AElf contracts are defined as services that are currently defined and generated with gRPC and protobuf.

As an example, here is part of the definition of our multi-token contract. Each functionality will be explained more in detail in their respective sections. Note that for simplicity, the contract has been simplified to show only the essential.

```
syntax = "proto3";

package token;
option csharp_namespace = "AElf.Contracts.MultiToken.Messages";

service TokenContract {
    option (aelf.csharp_state) = "AElf.Contracts.MultiToken.TokenContractState";

    // Actions
    rpc Create (CreateInput) returns (google.protobuf.Empty) { }
    rpc Transfer (TransferInput) returns (google.protobuf.Empty) { }

    // Views
    rpc GetBalance (GetBalanceInput) returns (GetBalanceOutput) {
        option (aelf.is_view) = true;
    }
}
```

For the service we have two different types of methods:

- Actions - these are normal smart contract methods that take input and output and usually modify the state of the chain.
- Views - these methods are special in the sense that they do not modify the state of the chain. They are usually used in some way to query the value of the contracts state.

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

The services takes a protobuf message as input and also returns a protobuf message as output. Note that here it returns a special message - google.protobuf.Empty - that signifies returning nothing. As a convention we append Input to any protobuf type that is destined to be a parameter to a service.

16.2.1 View option

```
rpc GetBalance (GetBalanceInput) returns (GetBalanceOutput) {
    option (aelf.is_view) = true;
}
```

This service is annotated with a view option. This signifies that this is a readonly method and will not modify the state.

16.3 Smart contract events

16.3.1 Event option

During the execution, Events are used internally to represent events that have happened during the execution of a smart contract. The event will be logged in the transaction traces logs (a collection of LogEvents).

```
message Transferred {
    option (aelf.is_event) = true;
    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;
    string memo = 5;
}
```

Notice the `option (aelf.is_event) = true;` line which indicates that the **Transferred** message is destined to be an event.

The following code demonstrates how to fire the event in a contract:

```
Context.Fire(new Transferred()
{
    From = from,
    To = to,
    ...
});
```

External code to the contract can monitor this after the execution of the transaction.

16.4 Smart contract messages

Here we define the concept of the message as defined by the protobuf language. We heavily use these messages to call smart contracts and serializing their state. The following is the definition of a simple message:

```
message CreateInput {
    string symbol = 1;
    sint64 totalSupply = 2;
    sint32 decimals = 3;
}
```

Here we see a message with three fields of type string, sint64 and sint32. In the message, you can use any type supported by protobuf, including composite messages, where one of your messages contains another message.

For message and service definitions, we use the **proto3** version of the protobuf language. You probably won't need to use most of the features that are provided, but here's the full reference for the language.

16.5 Development Requirements and Restrictions

There are several requirements and restrictions for a contract to be deployable that are classified into below categories:

16.5.1 Contract Project Requirements

Project Properties

- It is required to add `ContractCode` property in your contract project, so that the contract's DLL will be post processed by AElf's contract patcher to perform necessary injections that are required by code checks during deployment. Otherwise, deployment will fail.

```
<PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
  <RootNamespace>AElf.Contracts.MyContract</RootNamespace>
  <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
</PropertyGroup>

<PropertyGroup>
        <ContractCode Include="..\..\protobuf\my_contract.proto">
            <Link>Protobuf\Proto\my_contract.proto</Link>
        </ContractCode>
</PropertyGroup>
```

- It is required to enable `CheckForOverflowUnderflow` for both Release and Debug mode so that your contract will use arithmetic operators that will throw `OverflowException` if there is any overflow. This is to ensure that execution will not continue in case of an overflow in your contract and result with unpredictable output.

```
<PropertyGroup Condition=" '$(Configuration)' == 'Debug' ">
  <CheckForOverflowUnderflow>true</CheckForOverflowUnderflow>
</PropertyGroup>

<PropertyGroup Condition=" '$(Configuration)' == 'Release' ">
  <CheckForOverflowUnderflow>true</CheckForOverflowUnderflow>
</PropertyGroup>
```

If your contract contains any unchecked arithmetic operators, deployment will fail.

16.5.2 Contract Class Structure

Below restrictions are put in place to simplify code checks during deployment:

- Only 1 inheritance is allowed from `ContractBase` which is generated by the contract plugin as a nested type in `ContractContainer` and only 1 inheritance will be allowed from `CSharpSmartContract`. If there are multiple inheritances from `ContractBase` or `CSharpSmartContract`, code deployment will fail.

- Only 1 inheritance will be allowed from `ContractState`. Similar to above, if there are multiple inheritance from `AElf.Sdk.ContractState`, code check will fail.
- The type inherited from `ContractState` should be the element type of `CSharpSmartContract` generic instance type, otherwise code check will fail.

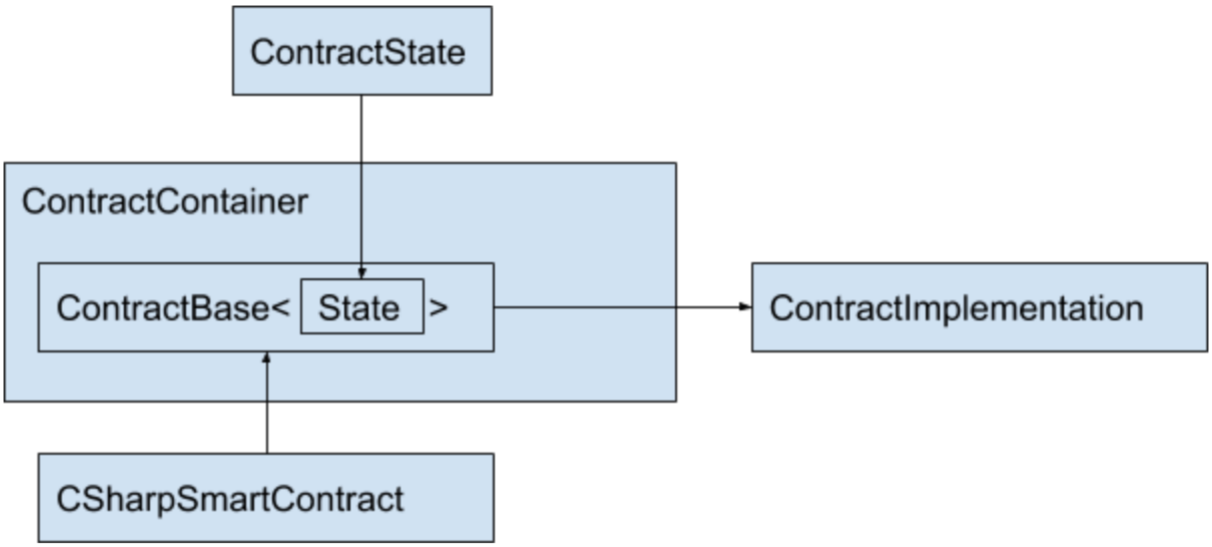


Fig. 1: Contract Class Structure

Limitations on Field Usage

In Contract Implementation Class

- Initial value for non-readonly, non-constant fields is not allowed. (Applied to all static / non-static fields) The reason is, their value will be reset to 0 or null after first execution and their initial value will be lost.

Allowed:

```
class MyContract : MyContractBase
{
  int test;
  static const int test = 2;
}
```

Not Allowed:

```
class MyContract : MyContractBase
{
!   int test = 2;
}
```

```
class MyContract : MyContractBase
{
  int test;

  public MyContract
```

(continues on next page)

(continued from previous page)

```
  {
!    test = 2;
  }
}
```

- Only primitive types, or one of below types are allowed for readonly / constant fields:

Type
All Primitive Types
`Marshaller<T>`
`Method<T, T>`
`MessageParser<T>`
`FieldCodec<T>`
`MapField<T, T>`
`ReadonlyCollection<T>`
`ReadonlyDictionary<T, T>`

* T can only be primitive type

In Non-Contract Classes (For classes that don't inherit from `ContractBase<T>`)

- Initial value for non-readonly, non-constant fields is not allowed for static fields. The reason is, their value will be reset to 0 or null after first execution and their initial value will be lost.

Allowed:

```
class AnyClass
{
  static int test;
}
```

Not Allowed:

```
class AnyClass
{
!  static int test = 2;
}
```

```
class AnyClass
{
  static int test;

  public AnyClass
  {
!    test = 2;
  }
}
```

Exception Case: Fields with FileDescriptor types. This is due to protobuf generated code. There are static fields `FileDescriptor` type fields generated by protobuf code and these fields don't have readonly modifier. We allow such fields only if they are FileDescriptor type and write access to these fields are only allowed from the constructor of the type where descriptor field is declared.

Allowed:

```
public class TestType
{
  private static FileDescriptor test;

  public class TestType
  {
    test = ...
  }
}
```

Not Allowed:

```
public class TestType
{
  private static FileDescriptor test;

  public TestType
  {
    test = ...
  }

!  public void SetFromSomeWhereElse(FileDescriptor input)
!  {
!    test = input;
!  }
}
```

Accessing to set `test` field is restricted to its declaring type's constructor only.

- Only below types are allowed for `readonly` / `constant` static fields:

Type
All Primitive Types
`Marshaller<T>`
`Method<T, T>`
`MessageParser<T>`
`FieldCodec<T>`
`MapField<T, T>`
`ReadonlyCollection<T>`
`ReadonlyDictionary<T, T>`

* T can only be primitive type

Exception Case: If a type has a `readonly` field same type as itself, it is only allowed if the type has no instance field.

This is to support Linq related generated types.

Allowed:

```
public class TestType
{
  private static readonly TestType test;

  private static int i;
}
```

Not Allowed:

```
public class TestType
{
  private static readonly TestType test;

!  private int i;
}
```

In Contract State

In contract state, only below types are allowed:

Primitive Types
BoolState
Int32State
UInt32State
Int64State
UInt64State
StringState
BytesState

Complex Types
SingletonState<T>
ReadonlyState<T>
MappedState<T, T>
MappedState<T, T, T>
MappedState<T, T, T, T>
MappedState<T, T, T, T, T>
MethodReference<T, T>
ProtobufState<T>
ContractReferenceState

16.5.3 Type and Namespace Restrictions

Nodes checks new contract code against below whitelist and if there is a usage of any type that is not covered in the whitelist, or the method access or type name is denied in below whitelist, the deployment will fail.

Assembly Dependencies

Assembly	Trust
netstandard.dll	Partial
System.Runtime.dll	Partial
System.Runtime.Extensions.dll	Partial
System.Private.CoreLib.dll	Partial
System.ObjectModel.dll	Partial
System.Linq.dll	Full
System.Collections	Full
Google.Protobuf.dll	Full
AElf.Sdk.CSharp.dll	Full
AElf.Types.dll	Full
AElf.CSharp.Core.dll	Full
AElf.Cryptography.dll	Full

Types and Members Whitelist in System Namespace

Type	Member (Field / Method)	Allowed
`Array`	`AsReadOnly`	Allowed
`Func<T>`	ALL	Allowed
`Func<T,T>`	ALL	Allowed
`Func<T,T,T>`	ALL	Allowed
`Nullable<T>`	ALL	Allowed
`Environment`	`CurrentManagedThreadId`	Allowed
`BitConverter`	`GetBytes`	Allowed
`NotImplementedException`	ALL	Allowed
`NotSupportedException`	ALL	Allowed
`ArgumentOutOfRangeException`	ALL	Allowed
`DateTime`	Partially	Allowed
`DateTime`	`Now`, `UtcNow`, `Today`	Denied
`Uri`	`TryCreate`	Allowed
`Uri`	`Scheme`	Allowed
`Uri`	`UriSchemeHttp`	Allowed
`Uri`	`UriSchemeHttps`	Allowed
`void`	ALL	Allowed
`object`	ALL	Allowed
`Type`	ALL	Allowed
`IDisposable`	ALL	Allowed
`Convert`	ALL	Allowed
`Math`	ALL	Allowed
`bool`	ALL	Allowed
`byte`	ALL	Allowed
`sbyte`	ALL	Allowed
`char`	ALL	Allowed
`int`	ALL	Allowed
`uint`	ALL	Allowed
`long`	ALL	Allowed
`ulong`	ALL	Allowed

Continued on next page

Table 1 – continued from previous page

Type	Member (Field / Method)	Allowed
decimal	ALL	Allowed
string	ALL	Allowed
string	Constructor	Denied
Byte[]	ALL	Allowed

Types and Members Whitelist in System.Reflection Namespace

Type	Member (Field / Method)	Allowed
AssemblyCompanyAttribute	ALL	Allowed
AssemblyConfigurationAttribute	ALL	Allowed
AssemblyFileVersionAttribute	ALL	Allowed
AssemblyInformationalVersionAttribute	ALL	Allowed
AssemblyProductAttribute	ALL	Allowed
AssemblyTitleAttribute	ALL	Allowed

Other Whitelisted Namespaces

Namespace	Type	Member	Allowed
System.Linq	ALL	ALL	Allowed
System.Collections	ALL	ALL	Allowed
System.Collections.Generic	ALL	ALL	Allowed
System.Collections.ObjectModel	ALL	ALL	Allowed
System.Globalization	CultureInfo	InvariantCulture	Allowed
System.Runtime.CompilerServices	RuntimeHelpers	InitializeArray	Allowed
System.Text	Encoding	UTF8,GetByteCount	Allowed

Allowed Types for Arrays

Type	Array Size Limit
byte	40960
short	20480
int	10240
long	5120
ushort	20480
uint	10240
ulong	5120
decimal	2560
char	20480
string	320
Type	5
Object	5
FileDescriptor	10
GeneratedClrTypeInfo	100

16.5.4 Other Restrictions

GetHashCode Usage

- *GetHashCode* method is only allowed to be called within *GetHashCode* methods. Calling *GetHashCode* methods from other methods is not allowed. This allows developers to implement their custom GetHashCode methods for their self defined types if required, and also allows protobuf generated message types.
- It is not allowed to set any field within *GetHashCode* methods.

Execution observer

- AElf's contract patcher will patch method call count observer for your contract. This is used to prevent infinitely method call like recursion. The number of method called in your contract will be counted during transaction execution. The observer will pause transaction execution if the number exceeds 15,000. The limit adjustment is governed by `Parliament`.
- AElf's contract patcher will patch method branch count observer for your contract. This is used to prevent infinitely loop case. The number of code control transfer in your contract will be counted during transaction execution. The observer will pause transaction execution if the number exceeds 15,000. The limit adjustment is governed by `Parliament`. The control transfer opcodes in C# contract are shown as below.

Opcode
`OpCodes.Beq`
`OpCodes.Beq_S`
`OpCodes.Bge`
`OpCodes.Bge_S`
`OpCodes.Bge_Un`
`OpCodes.Bge_Un_S`
`OpCodes.Bgt`
`OpCodes.Bgt_S`
`OpCodes.Ble`
`OpCodes.Ble_S`
`OpCodes.Ble_Un`
`OpCodes.Blt`
`OpCodes.Bne_Un`
`OpCodes.Bne_Un_S`
`OpCodes.Br`
`OpCodes.Brfalse`
`OpCodes.Brfalse_S`
`OpCodes.Brtrue`
`OpCodes.Brtrue`
`OpCodes.Brtrue_S`
`OpCodes.Br_S`

State size limit

- The size of data written to `State` would be limited every time. AElf's contract patcher is going to patch the code to validate your contract. As a result, you cannot write too big thing to contract and the limit is 128k by default. The limit adjustment is governed by `Parliament`.

CHAPTER 17

AELF API 1.0

17.1 Chain API

17.1.1 Get information about a given block by block hash. Optionally with the list of its transactions.

```
GET /api/blockChain/block
```

Parameters

Type	Name	Description	Schema	Default
Query	**blockHash** *optional*	block hash	string	
Query	**include Transactions** *optional*	include transactions or not	boolean	`"false"`

Responses

HTTP Code	Description	Schema
200	Success	*BlockDto*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`

- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.2 Get information about a given block by block height. Optionally with the list of its transactions.

```
GET /api/blockChain/blockByHeight
```

Parameters

Type	Name	Description	Schema	Default
Query	**blockHeight** *optional*	block height	integer (int64)	
Query	**include Transactions** *optional*	include transactions or not	boolean	`"false"`

Responses

HTTP Code	Description	Schema
200	Success	*BlockDto*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.3 Get the height of the current chain.

```
GET /api/blockChain/blockHeight
```

Responses

HTTP Code	Description	Schema
200	Success	integer (int64)

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.4 Get the current state about a given block

```
GET /api/blockChain/blockState
```

Parameters

Type	Name	Description	Schema
Query	**blockHash** *optional*	block hash	string

Responses

HTTP Code	Description	Schema
200	Success	*BlockStateDto*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.5 Get the current status of the block chain.

```
GET /api/blockChain/chainStatus
```

Responses

HTTP Code	Description	Schema
200	Success	*ChainStatusDto*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.6 Get the protobuf definitions related to a contract

```
GET /api/blockChain/contractFileDescriptorSet
```

Parameters

Type	Name	Description	Schema
Query	**address** *optional*	contract address	string

Responses

HTTP Code	Description	Schema
200	Success	string (byte)

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.7 POST /api/blockChain/executeRawTransaction

Parameters

Type	Name	Schema
Body	**input** *optional*	*ExecuteRawTransactionDto*

Responses

HTTP Code	Description	Schema
200	Success	string

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.8 Call a read-only method on a contract.

```
POST /api/blockChain/executeTransaction
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*ExecuteTransactionDto*

Responses

HTTP Code	Description	Schema
200	Success	string

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.9 Get the merkle path of a transaction.

```
GET /api/blockChain/merklePathByTransactionId
```

Parameters

Type	Name	Schema
Query	**transactionId** *optional*	string

Responses

HTTP Code	Description	Schema
200	Success	*MerklePathDto*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.10 Creates an unsigned serialized transaction

```
POST /api/blockChain/rawTransaction
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*CreateRawTransactionInput*

Responses

HTTP Code	Description	Schema
200	Success	*CreateRawTransactionOutput*

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.11 send a transaction

```
POST /api/blockChain/sendRawTransaction
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*SendRawTransactionInput*

Responses

HTTP Code	Description	Schema
200	Success	*SendRawTransactionOutput*

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.12 Broadcast a transaction

```
POST /api/blockChain/sendTransaction
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*SendTransactionInput*

Responses

HTTP Code	Description	Schema
200	Success	*SendTransactionOutput*

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.13 Broadcast multiple transactions

```
POST /api/blockChain/sendTransactions
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*SendTransactionsInput*

Responses

HTTP Code	Description	Schema
200	Success	< string > array

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.14 Estimate transaction fee

```
POST /api/blockChain/calculateTransactionFee
```

Parameters

Type	Name	Schema	Default
Body	**Input** *optional*	*CalculateTransactionFeeInput*	

Responses

HTTP Code	Description	Schema
200	Success	*CalculateTransactionFeeOutput*

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.15 GET /api/blockChain/taskQueueStatus

Responses

HTTP Code	Description	Schema
200	Success	< *TaskQueueInfoDto* > array

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.16 Get the transaction pool status.

```
GET /api/blockChain/transactionPoolStatus
```

Responses

HTTP Code	Description	Schema
200	Success	*GetTransactionPoolStatusOutput*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.17 Get the current status of a transaction

```
GET /api/blockChain/transactionResult
```

Parameters

Type	Name	Description	Schema
Query	**transactionId** *optional*	transaction id	string

Responses

HTTP Code	Description	Schema
200	Success	*TransactionResultDto*

The transaction result DTO object returned contains the transaction that contains the parameter values used for the call. The node will return the byte array as a base64 encoded string if it can't decode it.

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`

- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.1.18 Get multiple transaction results.

```
GET /api/blockChain/transactionResults
```

Parameters

Type	Name	Description	Schema	Default
Query	**blockHash** *optional*	block hash	string	
Query	**limit** *optional*	limit	integer (int32)	`10`
Query	**offset** *optional*	offset	integer (int32)	`0`

Responses

HTTP Code	Description	Schema
200	Success	< *TransactionResultDto* > array

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.2 Net API

17.2.1 Get information about the node's connection to the network.

```
GET /api/net/networkInfo
```

Responses

HTTP Code	Description	Schema
200	Success	*GetNetworkInfoOutput*

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- Net

17.2.2 Attempts to add a node to the connected network nodes

```
POST /api/net/peer
```

Parameters

Type	Name	Schema
Body	**input** *optional*	*AddPeerInput*

Responses

HTTP Code	Description	Schema
200	Success	boolean
401	Unauthorized	

Security

- Basic Authentication

Consumes

- `application/json-patch+json; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/*+json; v=1.0`
- `application/x-protobuf; v=1.0`

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- Net

17.2.3 Attempts to remove a node from the connected network nodes

```
DELETE /api/net/peer
```

Parameters

Type	Name	Description	Schema
Query	**address** *optional*	ip address	string

Responses

HTTP Code	Description	Schema
200	Success	boolean
401	Unauthorized	

Security

- Basic Authentication

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- Net

17.2.4 Get peer info about the connected network nodes

```
GET /api/net/peers
```

Parameters

Type	Name	Schema	Default
Query	**withMetrics** *optional*	boolean	`"false"`

Responses

HTTP Code	Description	Schema
200	Success	< *PeerDto* > array

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- BlockChain

17.2.5 Definitions

AddPeerInput

Name	Description	Schema
Address *optional*	ip address	string

BlockBodyDto

Name	Schema
Transactions *optional*	< string > array
TransactionsCount *optional*	integer (int32)

BlockDto

Name	Schema
BlockHash *optional*	string
Body *optional*	*BlockBodyDto*
Header *optional*	*BlockHeaderDto*
BlockSize *optional*	integer (int32)

BlockHeaderDto

Name	Schema
Bloom *optional*	string
ChainId *optional*	string
Extra *optional*	string
Height *optional*	integer (int64)
MerkleTreeRootOfTransactions *optional*	string
MerkleTreeRootOfWorldState *optional*	string
MerkleTreeRootOfTransactionState *optional*	string
PreviousBlockHash *optional*	string
SignerPubkey *optional*	string
Time *optional*	string (date-time)

BlockStateDto

Name	Schema
BlockHash *optional*	string
BlockHeight *optional*	integer (int64)
Changes *optional*	< string, string > map
Deletes *optional*	< string > array
PreviousHash *optional*	string

ChainStatusDto

Name	Schema
BestChainHash *optional*	string
BestChainHeight *optional*	integer (int64)
Branches *optional*	< string, integer (int64) > map
ChainId *optional*	string
GenesisBlockHash *optional*	string
GenesisContractAddress *optional*	string
LastIrreversibleBlockHash *optional*	string
LastIrreversibleBlockHeight *optional*	integer (int64)
LongestChainHash *optional*	string
LongestChainHeight *optional*	integer (int64)
NotLinkedBlocks *optional*	< string, string > map

CreateRawTransactionInput

Name	Description	Schema
From *required*	from address	string
MethodName *required*	contract method name	string
Params *required*	contract method parameters	string
RefBlockHash *required*	refer block hash	string
RefBlockNumber *required*	refer block height	integer (int64)
To *required*	to address	string

CreateRawTransactionOutput

Name	Schema
RawTransaction *optional*	string

ExecuteRawTransactionDto

Name	Description	Schema
RawTransaction *optional*	raw transaction	string
Signature *optional*	signature	string

ExecuteTransactionDto

Name	Description	Schema
RawTransaction *optional*	raw transaction	string

GetNetworkInfoOutput

Name	Description	Schema
Connections *optional*	total number of open connections between this node and other nodes	integer (int32)
ProtocolVersion *optional*	network protocol version	integer (int32)
Version *optional*	node version	string

GetTransactionPoolStatusOutput

Name	Schema
Queued *optional*	integer (int32)
Validated *optional*	integer (int32)

LogEventDto

Name	Schema
Address *optional*	string
Indexed *optional*	< string > array
Name *optional*	string
NonIndexed *optional*	string

MerklePathDto

Name	Schema
MerklePathNodes *optional*	< *MerklePathNodeDto* > array

MerklePathNodeDto

Name	Schema
Hash *optional*	string
IsLeftChildNode *optional*	boolean

MinerInRoundDto

Name	Schema
ActualMiningTimes *optional*	< string (date-time) > array
ExpectedMiningTime *optional*	string (date-time)
ImpliedIrreversibleBlockHeight *optional*	integer (int64)
InValue *optional*	string
MissedBlocks *optional*	integer (int64)
Order *optional*	integer (int32)
OutValue *optional*	string
PreviousInValue *optional*	string
ProducedBlocks *optional*	integer (int64)
ProducedTinyBlocks *optional*	integer (int32)

PeerDto

Name	Schema
BufferedAnnouncementsCount *optional*	integer (int32)
BufferedBlocksCount *optional*	integer (int32)
BufferedTransactionsCount *optional*	integer (int32)
ConnectionTime *optional*	integer (int64)
Inbound *optional*	boolean
IpAddress *optional*	string
ProtocolVersion *optional*	integer (int32)
RequestMetrics *optional*	< *RequestMetric* > array
ConnectionStatus *optional*	string
NodeVersion *optional*	string

RequestMetric

Name	Schema
Info *optional*	string
MethodName *optional*	string
RequestTime *optional*	*Timestamp*
RoundTripTime *optional*	integer (int64)

RoundDto

Name	Schema
Co nfirmedIrreversibleBlockHeight *optional*	integer (int64)
Confirm edIrreversibleBlockRoundNumber *optional*	integer (int64)
Ext raBlockProducerOfPreviousRound *optional*	string
IsMinerListJustChanged *optional*	boolean
RealTimeMinerInformation *optional*	< string, *MinerInRoundDto* > map
RoundId *optional*	integer (int64)
RoundNumber *optional*	integer (int64)
TermNumber *optional*	integer (int64)

SendRawTransactionInput

Name	Description	Schema
ReturnTransaction *optional*	return transaction detail or not	boolean
Signature *optional*	signature	string
Transaction *optional*	raw transaction	string

SendRawTransactionOutput

Name	Schema
Transaction *optional*	*TransactionDto*
TransactionId *optional*	string

SendTransactionInput

Name	Description	Schema
RawTransaction *optional*	raw transaction	string

SendTransactionOutput

Name	Schema
TransactionId *optional*	string

SendTransactionsInput

Name	Description	Schema
RawTransactions *optional*	raw transactions	string

TaskQueueInfoDto

Name	Schema
Name *optional*	string
Size *optional*	integer (int32)

Timestamp

Name	Schema
Nanos *optional*	integer (int32)
Seconds *optional*	integer (int64)

TransactionDto

Name	Schema
From *optional*	string
MethodName *optional*	string
Params *optional*	string
RefBlockNumber *optional*	integer (int64)
RefBlockPrefix *optional*	string
Signature *optional*	string
To *optional*	string

TransactionResultDto

Name	Schema
BlockHash *optional*	string
BlockNumber *optional*	integer (int64)
Bloom *optional*	string
Error *optional*	string
Logs *optional*	< *LogEventDto* > array
ReturnValue *optional*	string
Status *optional*	string
Transaction *optional*	*TransactionDto*
TransactionId *optional*	string
TransactionSize *optional*	integer (int32)

CalculateTransactionFeeInput

Name	Schema
RawTrasaction *optional*	string

CalculateTransactionFeeOutput

Name	Schema
Success *optional*	bool
TransactionFee *optional*	Dictionary<string, long>
ResourceFee *optional*	Dictionary<string, long>

CHAPTER 18

Chain SDK

18.1 aelf-sdk.js - AELF JavaScript API

18.1.1 Introduction

aelf-sdk.js for aelf is like web.js for ethereum.

aelf-sdk.js is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.js, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.js

18.1.2 Adding aelf-sdk.js

First you need to get aelf-sdk.js into your project. This can be done using the following methods:

npm: `npm install aelf-sdk`

pure js: `link dist/aelf.umd.js`

After that you need to create a aelf instance and set a provider.

```
// in brower use: <script src="https://unpkg.com/aelf-sdk@lastest/dist/aelf.umd.js"></
↪script>
// in node.js use: const AElf = require('aelf-sdk');
const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:8000'));
```

18.1.3 Examples

You can also see full examples in `./examples`;

Create instance

Create a new instance of AElf, connect to an AELF chain node.

```
import AElf from 'aelf-sdk';

// create a new instance of AElf
const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:1235'));
```

Create or load a wallet

Create or load a wallet with `AElf.wallet`

```
```javascript
// create a new wallet
const newWallet = AElf.wallet.createNewWallet();
// load a wallet by private key
const priviteKeyWallet = AElf.wallet.getWalletByPrivateKey('xxxxxxx');
// load a wallet by mnemonic
const mnemonicWallet = AElf.wallet.getWalletByMnemonic('set kite ...');
```
```

3.Get a system contract address

Get a system contract address, take `AElf.ContractNames.Token` as an example

```
const tokenContractName = 'AElf.ContractNames.Token';
let tokenContractAddress;
(async () => {
  // get chain status
  const chainStatus = await aelf.chain.getChainStatus();
  // get genesis contract address
  const GenesisContractAddress = chainStatus.GenesisContractAddress;
  // get genesis contract instance
  const zeroContract = await aelf.chain.contractAt(GenesisContractAddress,
↪newWallet);
  // Get contract address by the read only method `GetContractAddressByName` of
↪genesis contract
  tokenContractAddress = await zeroContract.GetContractAddressByName.call(AElf.
↪utils.sha256(tokenContractName));
})()
```

4.Get a contract instance

Get a contract instance by contract address

```
const wallet = AElf.wallet.createNewWallet();
let tokenContract;
// Use token contract for examples to demonstrate how to get a contract instance
↪in different ways
// in async function
(async () => {
  tokenContract = await aelf.chain.contractAt(tokenContractAddress, wallet)
```

(continues on next page)

(continued from previous page)

```
})();

// promise way
aelf.chain.contractAt(tokenContractAddress, wallet)
  .then(result => {
    tokenContract = result;
  });

// callback way
aelf.chain.contractAt(tokenContractAddress, wallet, (error, result) => {if␣
↪(error) throw error; tokenContract = result;});
```

5.Use contract instance

How to use contract instance

```
A contract instance consists of several contract methods and methods can be called in␣
↪two ways: read-only and send transaction.
```

```
(async () => {
  // get the balance of an address, this would not send a transaction,
  // or store any data on the chain, or required any transaction fee, only get␣
↪the balance
  // with `.call` method, `aelf-sdk` will only call read-only method
  const result = await tokenContract.GetBalance.call({
    symbol: "ELF",
    owner: "7s4XoUHfPuqoZAwnTV7pHWZAaivMiL8aZrDSnY9brE1woa8vz"
  });
  console.log(result);
  /**
  {
    "symbol": "ELF",
    "owner": "2661mQaaPnzLCoqXPeys3Vzf2wtGM1kSrqVBgNY4JUaGBxEsX8",
    "balance": "1000000000000"
  }*/
  // with no `.call`, `aelf-sdk` will sign and send a transaction to the chain,␣
↪and return a transaction id.
  // make sure you have enough transaction fee `ELF` in your wallet
  const transactionId = await tokenContract.Transfer({
    symbol: "ELF",
    to: "7s4XoUHfPuqoZAwnTV7pHWZAaivMiL8aZrDSnY9brE1woa8vz",
    amount: "1000000000",
    memo: "transfer in demo"
  });
  console.log(transactionId);
  /**
    {
      "TransactionId": "123123"
    }
  */
})()
```

6.Change the node endpoint

Change the node endpoint by using `aelf.setProvider`

```
```javascript
import AElf from 'aelf-sdk';

const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:1235'));
aelf.setProvider(new AElf.providers.HttpProvider('http://127.0.0.1:8000'));
```
```

18.1.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

parameters and returns based on the URL: `https://aelf-public-node.aelf.io/swagger/index.html`

The usage of these methods is based on the AElf instance, so if you don't have one please create it:

```javascript
import AElf from 'aelf-sdk';

// create a new instance of AElf, change the URL if needed
const aelf = new AElf(new AElf.providers.HttpProvider('http://127.0.0.1:1235'));
```

1.getChainStatus

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`Object`

- `ChainId - String`
- `Branches - Object`
- `NotLinkedBlocks - Object`
- `LongestChainHeight - Number`
- `LongestChainHash - String`
- `GenesisBlockHash - String`
- `GenesisContractAddress - String`
- `LastIrreversibleBlockHash - String`
- `LastIrreversibleBlockHeight - Number`
- `BestChainHash - String`

- `BestChainHeight - Number`

Example

```
aelf.chain.getChainStatus()
.then(res => {
  console.log(res);
})
```

2.getContractFileDescriptorSet

Get the protobuf definitions related to a contract

Web API path

`/api/blockChain/contractFileDescriptorSet`

Parameters

1. `contractAddress - String` address of a contract

Returns

`String`

Example

```
aelf.chain.getContractFileDescriptorSet(contractAddress)
  .then(res => {
    console.log(res);
  })
```

3.getBlockHeight

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`Number`

Example

```
aelf.chain.getBlockHeight()
  .then(res => {
    console.log(res);
  })
```

4.getBlock

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `blockHash - String`
2. `includeTransactions - Boolean :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`Object`

- `BlockHash - String`
- `Header - Object`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - Array`
  - `Height - Number`
  - `Time - google.protobuf.Timestamp`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - Object`
  - `TransactionsCount - Number`
  - `Transactions - Array`
    - `transactionId - String`

Example

```
aelf.chain.getBlock(blockHash, false)
  .then(res => {
    console.log(res);
  })
```

5.getBlockByHeight

Web API path

`/api/blockChain/blockByHeight`

Get block information by block height.

Parameters

1. `blockHeight - Number`
2. `includeTransactions - Boolean :`

- `true` require transaction ids list in the block

- `false` Doesn't require transaction ids list in the block

Returns

`Object`

- `BlockHash - String`
- `Header - Object`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - Array`
  - `Height - Number`
  - `Time - google.protobuf.Timestamp`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - Object`
  - `TransactionsCount - Number`
  - `Transactions - Array`
    - `transactionId - String`

Example

```
aelf.chain.getBlockByHeight(12, false)
  .then(res => {
    console.log(res);
  })
```

6.getTxResult

Get the result of a transaction

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - String`

Returns

`Object`

- `TransactionId - String`
- `Status - String`
- `Logs - Array`
  - `Address - String`
  - `Name - String`

- Indexed - Array
- NonIndexed - String

- Bloom - String
- BlockNumber - Number
- Transaction - Object
  - From - String
  - To - String
  - RefBlockNumber - Number
  - RefBlockPrefix - String
  - MethodName - String
  - Params - Object
  - Signature - String
- ReadableReturnValue - Object
- Error - String

Example

```
aelf.chain.getTxResult(transactionId)
  .then(res => {
    console.log(res);
  })
```

7.getTxResults

Get multiple transaction results in a block

Web API path

/api/blockChain/transactionResults

Parameters

1. blockHash - String
2. offset - Number
3. limit - Number

Returns Array - The array of method descriptions:

- the transaction result object

Example

```
aelf.chain.getTxResults(blockHash, 0, 2)
  .then(res => {
    console.log(res);
  })
```

8.getTransactionPoolStatus

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Parameters

Empty

9.sendTransaction

Broadcast a transaction

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`Object` - Serialization of data into protobuf data, The object with the following structure :

- `RawTransaction - String:`

usually developers don't need to use this function directly, just get a contract method and send transaction by call contract method:

10.sendTransactions

Broadcast multiple transactions

POST

Parameters

`Object` - The object with the following structure :

- `RawTransaction - String`

11.callReadOnly

Call a read-only method on a contract.

POST

Parameters

`Object` - The object with the following structure :

- `RawTransaction - String`

12.getPeers

Get peer info about the connected network nodes

GET

Parameters

1. `withMetrics - Boolean`:

- `true` with metrics
- `false` without metrics

13.addPeer

Attempts to add a node to the connected network nodes

POST

Parameters

`Object` - The object with the following structure :

- `Address - String`

14.removePeer

Attempts to remove a node from the connected network nodes

DELETE

Parameters

1. `address - String`

15.calculateTransactionFee

Estimate transaction fee

POST

Parameters

`Object` - The object with the following structure :

- `RawTransaction - String`

16.networkInfo

Get information about the node's connection to the network

GET

Parameters

Empty

18.1.5 AElf.wallet

`AElf.wallet` is a static property of `AElf`.

Use the api to see detailed results

1.createNewWallet

Returns

`Object`

- `mnemonic - String`: mnemonic
- `BIP44Path - String`: m/purpose'/coin_type'/account'/change/address_index
- `childWallet - Object`: HD Wallet
- `keyPair - String`: The EC key pair generated by elliptic
- `privateKey - String`: private Key
- `address - String`: address

Example

```
import AElf from 'aelf-sdk';
const wallet = AElf.wallet.createNewWallet();
```

2.getWalletByMnemonic

Parameters

1. `mnemonic - String` : wallet's mnemonic

Returns

`Object`: Complete wallet object.

Example

```
const wallet = AElf.wallet.getWalletByMnemonic(mnemonic);
```

3.getWalletByPrivateKey

Parameters

1. `privateKey: String` : wallet's private key

Returns

`Object`: Complete wallet object, with empty mnemonic

Example

```
const wallet = AElf.wallet.getWalletByPrivateKey(privateKey);
```

4.signTransaction

Use wallet `keypair` to sign a transaction

Parameters

1. `rawTxn - String`
2. `keyPair - String`

Returns

`Object`: The object with the following structure :

Example

```
const result = aelf.wallet.signTransaction(rawTxn, keyPair);
```

5.AESEncrypt

Encrypt a string by aes algorithm

Parameters

1. `input - String`
2. `password - String`

Returns

`String`

6.AESDecrypt

Decrypt by aes algorithm

Parameters

1. `input - String`
2. `password - String`

Returns

`String`

18.1.6 AElf.pbjs

The reference to protobuf.js, read the documentation to see how to use.

18.1.7 AElf.pbUtils

Some basic format methods of aelf.

For more information, please see the code in `src/utils/proto.js`. It is simple and easy to understand.

AElf.utils

Some methods for aelf.

For more information, please see the code in `src/utils/utils.js`. It is simple and easy to understand.

Check address

```
const AElf = require('aelf-sdk');
const {base58} = AElf.utils;
base58.decode('$addresss'); // throw error if invalid
```

18.1.8 AElf.version

```
import AElf from 'aelf-sdk';
AElf.version // eg. 3.2.23
```

18.1.9 Requirements

- Node.js
- NPM

18.1.10 Support

18.1.11 About contributing

Read out [contributing guide]

18.1.12 About Version

https://semver.org/

18.2 aelf-sdk.cs - AELF C# API

This C# library helps in the communication with an AElf node. You can find out more **here**.

18.2.1 Introduction

aelf-sdk.cs is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.cs, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.cs

18.2.2 Adding aelf-sdk.cs package

First you need to get AElf.Client package into your project. This can be done using the following methods:

Package Manager:

```
PM> Install-Package AElf.Client
```

.NET CLI

```
> dotnet add package AElf.Client
```

PackageReference

```
<PackageReference Include="AElf.Client" Version="X.X.X" />
```

18.2.3 Examples

Create instance

Create a new instance of AElfClient, and set url of an AElf chain node.

```
using AElf.Client.Service;

// create a new instance of AElfClient
AElfClient client = new AElfClient("http://127.0.0.1:1235");
```

Test connection

Check that the AElf chain node is connectable.

```
var isConnected = await client.IsConnectedAsync();
```

Initiate a transfer transaction

```
// Get token contract address.
var tokenContractAddress = await client.GetContractAddressByNameAsync(HashHelper.
→ComputeFrom("AElf.ContractNames.Token"));

var methodName = "Transfer";
var param = new TransferInput
{
    To = new Address {Value = Address.FromBase58(
→"7s4XoUHfPuqoZAwnTV7pHWZAaivMiL8aZrDSnY9brE1woa8vz").Value},
    Symbol = "ELF",
    Amount = 1000000000,
    Memo = "transfer in demo"
};
var ownerAddress = client.GetAddressFromPrivateKey(PrivateKey);
```

(continues on next page)

(continued from previous page)

```
// Generate a transfer transaction.
var transaction = await client.GenerateTransaction(ownerAddress, tokenContractAddress.
↪ToBase58(), methodName, param);
var txWithSign = client.SignTransaction(PrivateKey, transaction);

// Send the transfer transaction to AElf chain node.
var result = await client.SendTransactionAsync(new SendTransactionInput
{
    RawTransaction = txWithSign.ToByteArray().ToHex()
});

await Task.Delay(4000);
// After the transaction is mined, query the execution results.
var transactionResult = await client.GetTransactionResultAsync(result.TransactionId);
Console.WriteLine(transactionResult.Status);

// Query account balance.
var paramGetBalance = new GetBalanceInput
{
    Symbol = "ELF",
    Owner = new Address {Value = Address.FromBase58(ownerAddress).Value}
};
var transactionGetBalance =await client.GenerateTransaction(ownerAddress,
↪tokenContractAddress.ToBase58(), "GetBalance", paramGetBalance);
var txWithSignGetBalance = client.SignTransaction(PrivateKey, transactionGetBalance);

var transactionGetBalanceResult = await client.ExecuteTransactionAsync(new
↪ExecuteTransactionDto
{
    RawTransaction = txWithSignGetBalance.ToByteArray().ToHex()
});

var balance = GetBalanceOutput.Parser.ParseFrom(ByteArrayHelper.
↪HexstringToByteArray(transactionGetBalanceResult));
Console.WriteLine(balance.Balance);
```

18.2.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

The usage of these methods is based on the AElfClient instance, so if you don't have one please create it:

```
using AElf.Client.Service;

// create a new instance of AElf, change the URL if needed
AElfClient client = new AElfClient("http://127.0.0.1:1235");
```

GetChainStatus

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`ChainStatusDto`

- `ChainId - string`
- `Branches - Dictionary<string,long>`
- `NotLinkedBlocks - Dictionary<string,string>`
- `LongestChainHeight - long`
- `LongestChainHash - string`
- `GenesisBlockHash - string`
- `GenesisContractAddress - string`
- `LastIrreversibleBlockHash - string`
- `LastIrreversibleBlockHeight - long`
- `BestChainHash - string`
- `BestChainHeight - long`

Example

```
await client.GetChainStatusAsync();
```

GetContractFileDescriptorSet

Get the protobuf definitions related to a contract.

Web API path

`/api/blockChain/contractFileDescriptorSet`

Parameters

1. `contractAddress - string` address of a contract

Returns

`byte[]`

Example

```
await client.GetContractFileDescriptorSetAsync(address);
```

GetBlockHeight

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`long`

Example

```
await client.GetBlockHeightAsync();
```

GetBlock

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `blockHash - string`
2. `includeTransactions - bool :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - string`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - string`
  - `MerkleTreeRootOfTransactions - string`
  - `MerkleTreeRootOfWorldState - string`
  - `Extra - string`
  - `Height - long`
  - `Time - DateTime`
  - `ChainId - string`
  - `Bloom - string`
  - `SignerPubkey - string`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`
  - `Transactions - List<string>`

Example

```
await client.GetBlockByHashAsync(blockHash);
```

GetBlockByHeight

Web API path

`/api/blockChain/blockByHeight`

Get block information by block height.

Parameters

1. `blockHeight - long`
2. `includeTransactions - bool :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - string`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - string`
  - `MerkleTreeRootOfTransactions - string`
  - `MerkleTreeRootOfWorldState - string`
  - `Extra - string`
  - `Height - long`
  - `Time - DateTime`
  - `ChainId - string`
  - `Bloom - string`
  - `SignerPubkey - string`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`
  - `Transactions - List<string>`

Example

```
await client.GetBlockByHeightAsync(height);
```

GetTransactionResult

Get the result of a transaction

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - string`

Returns

`TransactionResultDto`

- `TransactionId - string`
- `Status - string`
- `Logs - LogEventDto[]`
  - `Address - string`
  - `Name - string`
  - `Indexed - string[]`
  - `NonIndexed - string`
- `Bloom - string`
- `BlockNumber - long`
- `Transaction - TransactionDto`
  - `From - string`
  - `To - string`
  - `RefBlockNumber - long`
  - `RefBlockPrefix - string`
  - `MethodName - string`
  - `Params - string`
  - `Signature - string`
- `Error - string`

Example

```
await client.GetTransactionResultAsync(transactionId);
```

GetTransactionResults

Get multiple transaction results in a block.

Web API path

`/api/blockChain/transactionResults`

Parameters

1. `blockHash - string`
2. `offset - int`
3. `limit - int`

Returns

`List<TransactionResultDto>` - The array of transaction result:

- the transaction result object

Example

```
await client.GetTransactionResultsAsync(blockHash, 0, 10);
```

GetTransactionPoolStatus

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Parameters

Empty

Returns

`TransactionPoolStatusOutput`

- `Queued` - int
- `Validated` - int

Example

```
await client.GetTransactionPoolStatusAsync();
```

SendTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendTransactionInput` - Serialization of data into protobuf data:

- `RawTransaction - string:`

Returns

`SendTransactionOutput`

- `TransactionId` - string

Example

```
await client.SendTransactionAsync(input);
```

SendRawTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendRawTransactionInput` - Serialization of data into protobuf data:

- `Transaction - string`
- `Signature - string`
- `ReturnTransaction - bool`

Returns

`SendRawTransactionOutput`

- `TransactionId - string`
- `Transaction - TransactionDto`

Example

```
await client.SendRawTransactionAsync(input);
```

SendTransactions

Broadcast multiple transactions.

Web API path

`/api/blockChain/sendTransactions`

POST

Parameters

`SendTransactionsInput` - Serialization of data into protobuf data:

- `RawTransactions - string`

Returns

`string[]`

Example

```
await client.SendTransactionsAsync(input);
```

CreateRawTransaction

Creates an unsigned serialized transaction.

Web API path

`/api/blockChain/rawTransaction`

POST

Parameters

`CreateRawTransactionInput`

- `From - string`
- `To - string`
- `RefBlockNumber - long`

- RefBlockHash - string
- MethodName - string
- Params - string

Returns

CreateRawTransactionOutput- Serialization of data into protobuf data:

- RawTransactions - string

Example

```
await client.CreateRawTransactionAsync(input);
```

ExecuteTransaction

Call a read-only method on a contract.

Web API path

/api/blockChain/executeTransaction

POST

Parameters

ExecuteTransactionDto - Serialization of data into protobuf data:

- RawTransaction - string

Returns

string

Example

```
await client.ExecuteTransactionAsync(input);
```

ExecuteRawTransaction

Call a read-only method on a contract.

Web API path

/api/blockChain/executeRawTransaction

POST

Parameters

ExecuteRawTransactionDto - Serialization of data into protobuf data:

- RawTransaction - string
- Signature - string

Returns

string

Example

```
await client.ExecuteRawTransactionAsync(input);
```

GetPeers

Get peer info about the connected network nodes.

Web API path

/api/net/peers

Parameters

1. withMetrics - bool

Returns

List<PeerDto>

- IpAddress - string
- ProtocolVersion - int
- ConnectionTime - long
- ConnectionStatus - string
- Inbound - bool
- BufferedTransactionsCount - int
- BufferedBlocksCount - int
- BufferedAnnouncementsCount - int
- RequestMetrics - List<RequestMetric>
  - RoundTripTime - long
  - MethodName - string
  - Info - string
  - RequestTime - string

Example

```
await client.GetPeersAsync(false);
```

AddPeer

Attempts to add a node to the connected network nodes.

Web API path

/api/net/peer

POST

Parameters

1. ipAddress - string

Returns

`bool`

Example

```
await client.AddPeerAsync("127.0.0.1:7001");
```

RemovePeer

Attempts to remove a node from the connected network nodes.

Web API path

`/api/net/peer`

DELETE

Parameters

1. `ipAddress - string`

Returns

`bool`

Example

```
await client.RemovePeerAsync("127.0.0.1:7001");
```

GetNetworkInfo

Get the network information of the node.

Web API path

`/api/net/networkInfo`

Parameters

Empty

Returns

`NetworkInfoOutput`

- `Version - string`
- `ProtocolVersion - int`
- `Connections - int`

Example

```
await client.GetNetworkInfoAsync();
```

18.2.5 AElf Client

IsConnected

Verify whether this sdk successfully connects the chain.

Parameters

Empty

Returns

`bool`

Example

```
await client.IsConnectedAsync();
```

GetGenesisContractAddress

Get the address of genesis contract.

Parameters

Empty

Returns

`string`

Example

```
await client.GetGenesisContractAddressAsync();
```

GetContractAddressByName

Get address of a contract by given contractNameHash.

Parameters

1. `contractNameHash - Hash`

Returns

`Address`

Example

```
await client.GetContractAddressByNameAsync(contractNameHash);
```

GenerateTransaction

Build a transaction from the input parameters.

Parameters

1. `from - string`
2. `to - string`

3. methodName - string
4. input - IMessage

Returns

Transaction

Example

```
await client.GenerateTransactionAsync(from, to, methodName, input);
```

GetFormattedAddress

Convert the Address to the displayed stringsymbol_base58-string_base58-string-chain-id.

Parameters

1. address - Address

Returns

string

Example

```
await client.GetFormattedAddressAsync(address);
```

SignTransaction

Sign a transaction using private key.

Parameters

1. privateKeyHex - string
2. transaction - Transaction

Returns

Transaction

Example

```
client.SignTransaction(privateKeyHex, transaction);
```

GetAddressFromPubKey

Get the account address through the public key.

Parameters

1. pubKey - string

Returns

string

Example

```
client.GetAddressFromPubKey(pubKey);
```

GetAddressFromPrivateKey

Get the account address through the private key.

Parameters

1. `privateKeyHex - string`

Returns

`string`

Example

```
client.GetAddressFromPrivateKey(privateKeyHex);
```

GenerateKeyPairInfo

Generate a new account key pair.

Parameters

Empty

Returns

`KeyPairInfo`

- `PrivateKey - string`
- `PublicKey - string`
- `Address - string`

Example

```
client.GenerateKeyPairInfo();
```

18.2.6 Supports

.NET Standard 2.0

18.3 aelf-sdk.go - AELF Go API

This Go library helps in the communication with an AElf node. You can find out more **here**.

18.3.1 Introduction

aelf-sdk.go is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.go, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.go

18.3.2 Adding aelf-sdk.go package

First you need to get aelf-sdk.go:

```
> go get -u github.com/AElfProject/aelf-sdk.go
```

18.3.3 Examples

Create instance

Create a new instance of AElfClient, and set url of an AElf chain node.

```
import ("github.com/AElfProject/aelf-sdk.go/client")

var aelf = client.AElfClient{
        Host:       "http://127.0.0.1:8000",
        Version:    "1.0",
        PrivateKey: "cd86ab6347d8e52bbbe8532141fc59ce596268143a308d1d40fedf385528b458
↪",
}
```

Initiate a transfer transaction

```
// Get token contract address.
tokenContractAddress, _ := aelf.GetContractAddressByName("AElf.ContractNames.Token")
fromAddress := aelf.GetAddressFromPrivateKey(aelf.PrivateKey)
methodName := "Transfer"
toAddress, _ := util.Base58StringToAddress(
↪"7s4XoUHfPuqoZAwnTV7pHWZAaivMiL8aZrDSnY9brE1woa8vz")

params := &pb.TransferInput{
        To:     toAddress,
        Symbol: "ELF",
        Amount: 1000000000,
        Memo:   "transfer in demo",
}
paramsByte, _ := proto.Marshal(params)

// Generate a transfer transaction.
transaction, _ := aelf.CreateTransaction(fromAddress, tokenContractAddress,␣
↪methodName, paramsByte)
signature, _ := aelf.SignTransaction(aelf.PrivateKey, transaction)
transaction.Signature = signature

// Send the transfer transaction to AElf chain node.
transactionByets, _ := proto.Marshal(transaction)
sendResult, _ := aelf.SendTransaction(hex.EncodeToString(transactionByets))

time.Sleep(time.Duration(4) * time.Second)
transactionResult, _ := aelf.GetTransactionResult(sendResult.TransactionID)
fmt.Println(transactionResult)
```

(continues on next page)

(continued from previous page)

```
// Query account balance.
ownerAddress, _ := util.Base58StringToAddress(fromAddress)
getBalanceInput := &pb.GetBalanceInput{
        Symbol: "ELF",
        Owner:  ownerAddress,
}
getBalanceInputByte, _ := proto.Marshal(getBalanceInput)

getBalanceTransaction, _ := aelf.CreateTransaction(fromAddress, tokenContractAddress,
↪"GetBalance", getBalanceInputByte)
getBalanceTransaction.Params = getBalanceInputByte
getBalanceSignature, _ := aelf.SignTransaction(aelf.PrivateKey, getBalanceTransaction)
getBalanceTransaction.Signature = getBalanceSignature

getBalanceTransactionByets, _ := proto.Marshal(getBalanceTransaction)
getBalanceResult, _ := aelf.ExecuteTransaction(hex.
↪EncodeToString(getBalanceTransactionByets))
balance := &pb.GetBalanceOutput{}
getBalanceResultBytes, _ := hex.DecodeString(getBalanceResult)
proto.Unmarshal(getBalanceResultBytes, balance)
fmt.Println(balance)
```

18.3.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

The usage of these methods is based on the AElfClient instance, so if you don't have one please create it:

```
import ("github.com/AElfProject/aelf-sdk.go/client")

var aelf = client.AElfClient{
        Host:       "http://127.0.0.1:8000",
        Version:    "1.0",
        PrivateKey: "680afd630d82ae5c97942c4141d60b8a9fedfa5b2864fca84072c17ee1f72d9d
↪",
}
```

GetChainStatus

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`ChainStatusDto`

- `ChainId - string`

- `Branches - map[string]interface{}`
- `NotLinkedBlocks - map[string]interface{}`
- `LongestChainHeight - int64`
- `LongestChainHash - string`
- `GenesisBlockHash - string`
- `GenesisContractAddress - string`
- `LastIrreversibleBlockHash - string`
- `LastIrreversibleBlockHeight - int64`
- `BestChainHash - string`
- `BestChainHeight - int64`

Example

```
chainStatus, err := aelf.GetChainStatus()
```

GetContractFileDescriptorSet

Get the protobuf definitions related to a contract.

Web API path

`/api/blockChain/contractFileDescriptorSet`

Parameters

1. `contractAddress - string` address of a contract

Returns

`byte[]`

Example

```
contractFile, err := aelf.GetContractFileDescriptorSet(
↪"pykr77ft9UUKJZLVq15wCH8PinBSjVRQ12sD1Ayq92mKFsJ1i")
```

GetBlockHeight

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`float64`

Example

```
height, err := aelf.GetBlockHeight()
```

GetBlock

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `blockHash - string`
2. `includeTransactions - bool :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - string`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - string`
  - `MerkleTreeRootOfTransactions - string`
  - `MerkleTreeRootOfWorldState - string`
  - `Extra - string`
  - `Height - int64`
  - `Time - string`
  - `ChainId - string`
  - `Bloom - string`
  - `SignerPubkey - string`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`
  - `Transactions - []string`

Example

```
block, err := aelf.GetBlockByHash(blockHash, true)
```

GetBlockByHeight

Web API path

`/api/blockChain/blockByHeight`

Get block information by block height.

Parameters

1. `blockHeight - int64`
2. `includeTransactions - bool`:

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - string`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - string`
  - `MerkleTreeRootOfTransactions - string`
  - `MerkleTreeRootOfWorldState - string`
  - `Extra - string`
  - `Height - int64`
  - `Time - string`
  - `ChainId - string`
  - `Bloom - string`
  - `SignerPubkey - string`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`
  - `Transactions - []string`

Example

```
block, err := aelf.GetBlockByHeight(100, true)
```

GetTransactionResult

Get the result of a transaction.

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - string`

Returns

`TransactionResultDto`

- `TransactionId - string`
- `Status - string`
- `Logs - []LogEventDto`
  - `Address - string`

   - Name - string
   - Indexed - []string
   - NonIndexed - string
- Bloom - string
- BlockNumber - int64
- BlockHash - string
- Transaction - TransactionDto
   - From - string
   - To - string
   - RefBlockNumber - int64
   - RefBlockPrefix - string
   - MethodName - string
   - Params - string
   - Signature - string
- ReturnValue - string
- Error - string

Example

```
transactionResult, err := aelf.GetTransactionResult(transactionID)
```

GetTransactionResults

Get multiple transaction results in a block.

Web API path

`/api/blockChain/transactionResults`

Parameters

1. blockHash - string
2. offset - int
3. limit - int

Returns

`[]TransactionResultDto` - The array of transaction result:

- the transaction result object

Example

```
transactionResults, err := aelf.GetTransactionResults(blockHash, 0, 10)
```

GetTransactionPoolStatus

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Parameters

Empty

Returns

`TransactionPoolStatusOutput`

- `Queued` - int
- `Validated` - int

Example

```
poolStatus, err := aelf.GetTransactionPoolStatus()
```

SendTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendTransactionInput` - Serialization of data into protobuf data:

- `RawTransaction - string`

Returns

`SendTransactionOutput`

- `TransactionId - string`

Example

```
sendResult, err := aelf.SendTransaction(input)
```

SendRawTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendRawTransactionInput` - Serialization of data into protobuf data:

- `Transaction - string`

- `Signature - string`
- `ReturnTransaction - bool`

Returns

`SendRawTransactionOutput`

- `TransactionId - string`
- `Transaction - TransactionDto`

Example

```
sendRawResult, err := aelf.SendRawTransaction(input)
```

SendTransactions

Broadcast multiple transactions.

Web API path

`/api/blockChain/sendTransactions`

POST

Parameters

`rawTransactions - string` - Serialization of data into protobuf data:

Returns

`[]interface{}`

Example

```
results, err := aelf.SendTransactions(transactions)
```

CreateRawTransaction

Creates an unsigned serialized transaction.

Web API path

`/api/blockChain/rawTransaction`

POST

Parameters

`CreateRawTransactionInput`

- `From - string`
- `To - string`
- `RefBlockNumber - int64`
- `RefBlockHash - string`
- `MethodName - string`
- `Params - string`

Returns

`CreateRawTransactionOutput`- Serialization of data into protobuf data:

- `RawTransactions - string`

Example

```
result, err := aelf.CreateRawTransaction(input)
```

ExecuteTransaction

Call a read-only method on a contract.

Web API path

`/api/blockChain/executeTransaction`

POST

Parameters

`rawTransaction - string`

Returns

`string`

Example

```
executeresult, err := aelf.ExecuteTransaction(rawTransaction)
```

ExecuteRawTransaction

Call a read-only method on a contract.

Web API path

`/api/blockChain/executeRawTransaction`

POST

Parameters

`ExecuteRawTransactionDto` - Serialization of data into protobuf data:

- `RawTransaction - string`
- `Signature - string`

Returns

`string`

Example

```
executeRawresult, err := aelf.ExecuteRawTransaction(executeRawinput)
```

GetPeers

Get peer info about the connected network nodes.

Web API path

`/api/net/peers`

Parameters

1. `withMetrics - bool`

Returns

`[]PeerDto`

- `IpAddress - string`
- `ProtocolVersion - int`
- `ConnectionTime - int64`
- `ConnectionStatus - string`
- `Inbound - bool`
- `BufferedTransactionsCount - int`
- `BufferedBlocksCount - int`
- `BufferedAnnouncementsCount - int`
- `RequestMetrics - []RequestMetric`
  - `RoundTripTime - int64`
  - `MethodName - string`
  - `Info - string`
  - `RequestTime - string`

Example

```
peers, err := aelf.GetPeers(false);
```

AddPeer

Attempts to add a node to the connected network nodes.

Web API path

`/api/net/peer`

POST

Parameters

1. `ipAddress - string`

Returns

`bool`

Example

```
addResult, err := aelf.AddPeer("127.0.0.1:7001");
```

RemovePeer

Attempts to remove a node from the connected network nodes.

Web API path

`/api/net/peer`

DELETE

Parameters

1. `ipAddress - string`

Returns

`bool`

Example

```
removeResult, err := aelf.RemovePeer("127.0.0.1:7001");
```

GetNetworkInfo

Get the network information of the node.

Web API path

`/api/net/networkInfo`

Parameters

Empty

Returns

`NetworkInfoOutput`

- `Version - string`
- `ProtocolVersion - int`
- `Connections - int`

Example

```
networkInfo, err := aelf.GetNetworkInfo()
```

18.3.5 AElf Client

IsConnected

Verify whether this sdk successfully connects the chain.

Parameters

Empty

Returns

`bool`

Example

```
isConnected := aelf.IsConnected()
```

GetGenesisContractAddress

Get the address of genesis contract.

Parameters

Empty

Returns

`string`

Example

```
contractAddress, err := aelf.GetGenesisContractAddress()
```

GetContractAddressByName

Get address of a contract by given contractNameHash.

Parameters

1. `contractNameHash - string`

Returns

`Address`

Example

```
contractAddress, err := aelf.GetContractAddressByName("AElf.ContractNames.Token")
```

CreateTransaction

Build a transaction from the input parameters.

Parameters

1. `from - string`
2. `to - string`
3. `methodName - string`
4. `params - []byte`

Returns

`Transaction`

Example

```
transaction, err := aelf.CreateTransaction(fromAddress, toAddress, methodName, param)
```

GetFormattedAddress

Convert the Address to the displayed stringsymbol_base58-string_base58-string-chain-id.

Parameters

1. `address - string`

Returns

`string`

Example

```
formattedAddress, err := aelf.GetFormattedAddress(address);
```

SignTransaction

Sign a transaction using private key.

Parameters

1. `privateKey - string`
2. `transaction - Transaction`

Returns

`[]byte`

Example

```
signature, err := aelf.SignTransaction(privateKey, transaction)
```

GetAddressFromPubKey

Get the account address through the public key.

Parameters

1. `pubKey - string`

Returns

`string`

Example

```
address := aelf.GetAddressFromPubKey(pubKey);
```

GetAddressFromPrivateKey

Get the account address through the private key.

Parameters

1. `privateKey - string`

Returns

`string`

Example

```
address := aelf.GetAddressFromPrivateKey(privateKey)
```

GenerateKeyPairInfo

Generate a new account key pair.

Parameters

Empty

Returns

`KeyPairInfo`

- `PrivateKey - string`
- `PublicKey - string`
- `Address - string`

Example

```
keyPair := aelf.GenerateKeyPairInfo()
```

18.3.6 Supports

Go 1.13

18.4 aelf-sdk.java - AELF Java API

This Java library helps in the communication with an AElf node. You can find out more **here**.

18.4.1 Introduction

aelf-sdk.java is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.java, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.java

18.4.2 Adding aelf-sdk.java package

First you need to get elf-sdk.java package into your project: MvnRepository

Maven:

```
<!-- https://mvnrepository.com/artifact/io.aelf/aelf-sdk -->
<dependency>
    <groupId>io.aelf</groupId>
    <artifactId>aelf-sdk</artifactId>
    <version>0.X.X</version>
</dependency>
```

18.4.3 Examples

Create instance

Create a new instance of AElfClient, and set url of an AElf chain node.

```
using AElf.Client.Service;

// create a new instance of AElf, change the URL if needed
AElfClient client = new AElfClient("http://127.0.0.1:1235");
```

Test connection

Check that the AElf chain node is connectable.

```
boolean isConnected = client.isConnected();
```

Initiate a transfer transaction

```
// Get token contract address.
String tokenContractAddress = client.getContractAddressByName(privateKey, Sha256.
→getBytesSha256("AElf.ContractNames.Token"));

Client.Address.Builder to = Client.Address.newBuilder();
to.setValue(ByteString.copyFrom(Base58.decodeChecked(
→"7s4XoUHfPuqoZAwnTV7pHWZAaivMiL8aZrDSnY9brE1woa8vz")));
Client.Address toObj = to.build();

TokenContract.TransferInput.Builder paramTransfer = TokenContract.TransferInput.
→newBuilder();
paramTransfer.setTo(toObj);
paramTransfer.setSymbol("ELF");
paramTransfer.setAmount(1000000000);
paramTransfer.setMemo("transfer in demo");
TokenContract.TransferInput paramTransferObj = paramTransfer.build();

String ownerAddress = client.getAddressFromPrivateKey(privateKey);

Transaction.Builder transactionTransfer = client.generateTransaction(ownerAddress,␣
→tokenContractAddress, "Transfer", paramTransferObj.toByteArray());
Transaction transactionTransferObj = transactionTransfer.build();
transactionTransfer.setSignature(ByteString.copyFrom(ByteArrayHelper.
→hexToByteArray(client.signTransaction(privateKey, transactionTransferObj))));
transactionTransferObj = transactionTransfer.build();
```

(continues on next page)

(continued from previous page)

```
// Send the transfer transaction to AElf chain node.
SendTransactionInput sendTransactionInputObj = new SendTransactionInput();
sendTransactionInputObj.setRawTransaction(Hex.toHexString(transactionTransferObj.
→toByteArray()));
SendTransactionOutput sendResult = client.sendTransaction(sendTransactionInputObj);

Thread.sleep(4000);
// After the transaction is mined, query the execution results.
TransactionResultDto transactionResult = client.getTransactionResult(sendResult.
→getTransactionId());
System.out.println(transactionResult.getStatus());

// Query account balance.
Client.Address.Builder owner = Client.Address.newBuilder();
owner.setValue(ByteString.copyFrom(Base58.decodeChecked(ownerAddress)));
Client.Address ownerObj = owner.build();

TokenContract.GetBalanceInput.Builder paramGetBalance = TokenContract.GetBalanceInput.
→newBuilder();
paramGetBalance.setSymbol("ELF");
paramGetBalance.setOwner(ownerObj);
TokenContract.GetBalanceInput paramGetBalanceObj = paramGetBalance.build();

Transaction.Builder transactionGetBalance = client.generateTransaction(ownerAddress,
→tokenContractAddress, "GetBalance", paramGetBalanceObj.toByteArray());
Transaction transactionGetBalanceObj = transactionGetBalance.build();
String signature = client.signTransaction(privateKey, transactionGetBalanceObj);
transactionGetBalance.setSignature(ByteString.copyFrom(ByteArrayHelper.
→hexToByteArray(signature)));
transactionGetBalanceObj = transactionGetBalance.build();

ExecuteTransactionDto executeTransactionDto = new ExecuteTransactionDto();
executeTransactionDto.setRawTransaction(Hex.toHexString(transactionGetBalanceObj.
→toByteArray()));
String transactionGetBalanceResult = client.executeTransaction(executeTransactionDto);

TokenContract.GetBalanceOutput balance = TokenContract.GetBalanceOutput.
→getDefaultInstance().parseFrom(ByteArrayHelper.
→hexToByteArray(transactionGetBalanceResult));
System.out.println(balance.getBalance());
```

18.4.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

The usage of these methods is based on the AElfClient instance, so if you don't have one please create it:

```
using AElf.Client.Service;

// create a new instance of AElf, change the URL if needed
AElfClient client = new AElfClient("http://127.0.0.1:1235");
```

GetChainStatus

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`ChainStatusDto`

- `ChainId - String`
- `Branches - HashMap<String, Long>`
- `NotLinkedBlocks - ashMap<String, String>`
- `LongestChainHeight - long`
- `LongestChainHash - String`
- `GenesisBlockHash - String`
- `GenesisContractAddress - String`
- `LastIrreversibleBlockHash - String`
- `LastIrreversibleBlockHeight - long`
- `BestChainHash - String`
- `BestChainHeight - long`

Example

```
client.getChainStatus();
```

GetContractFileDescriptorSet

Get the protobuf definitions related to a contract.

Web API path

`/api/blockChain/contractFileDescriptorSet`

Parameters

1. `contractAddress - String` address of a contract

Returns

`byte[]`

Example

```
client.getContractFileDescriptorSet(address);
```

GetBlockHeight

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`long`

Example

```
client.getBlockHeight();
```

GetBlock

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `blockHash - String`
2. `includeTransactions - boolean:`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - String`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - String`
  - `Height - long`
  - `Time - Date`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`

- Transactions - List<String>

Example

```
client.getBlockByHash(blockHash);
```

GetBlockByHeight

Web API path

`/api/blockChain/blockByHeight`

Get block information by block height.

Parameters

1. `blockHeight - long`
2. `includeTransactions - boolean :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`BlockDto`

- `BlockHash - String`
- `Header - BlockHeaderDto`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - String`
  - `Height - long`
  - `Time - Date`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - BlockBodyDto`
  - `TransactionsCount - int`
  - `Transactions - List<String>`

Example

```
client.getBlockByHeight(height);
```

GetTransactionResult

Get the result of a transaction.

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - String`

Returns

`TransactionResultDto`

- `TransactionId - String`
- `Status - String`
- `Logs - ist<LogEventDto>`
  - `Address - String`
  - `Name - String`
  - `Indexed - List<String>`
  - `NonIndexed - String`
- `Bloom - String`
- `BlockNumber - long`
- `Transaction - TransactionDto`
  - `From - String`
  - `To - String`
  - `RefBlockNumber - long`
  - `RefBlockPrefix - String`
  - `MethodName - String`
  - `Params - String`
  - `Signature - String`
- `Error - String`

Example

```
client.getTransactionResult(transactionId);
```

GetTransactionResults

Get multiple transaction results in a block.

Web API path

`/api/blockChain/transactionResults`

Parameters

1. `blockHash - String`

2. `offset - int`
3. `limit - int`

Returns

`List<TransactionResultDto>` - The array of transaction result:

- the transaction result object

Example

```
client.getTransactionResults(blockHash, 0, 10);
```

GetTransactionPoolStatus

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Parameters

Empty

Returns

`TransactionPoolStatusOutput`

- `Queued` - int
- `Validated` - int

Example

```
client.getTransactionPoolStatus();
```

SendTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendTransactionInput` - Serialization of data into protobuf data:

- `RawTransaction - String`

Returns

`SendTransactionOutput`

- `TransactionId - String`

Example

```
client.sendTransaction(input);
```

SendRawTransaction

Broadcast a transaction.

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`SendRawTransactionInput` - Serialization of data into protobuf data:

- `Transaction - String`
- `Signature - String`
- `ReturnTransaction - boolean`

Returns

`SendRawTransactionOutput`

- `TransactionId - String`
- `Transaction - TransactionDto`

Example

```
client.sendRawTransaction(input);
```

SendTransactions

Broadcast multiple transactions.

Web API path

`/api/blockChain/sendTransactions`

POST

Parameters

`SendTransactionsInput` - Serialization of data into protobuf data:

- `RawTransactions - String`

Returns

`List<String>`

Example

```
client.sendTransactions(input);
```

CreateRawTransaction

Creates an unsigned serialized transaction.

Web API path

`/api/blockChain/rawTransaction`

POST

Parameters

`CreateRawTransactionInput`

- `From - String`
- `To - String`
- `RefBlockNumber - long`
- `RefBlockHash - String`
- `MethodName - String`
- `Params - String`

Returns

`CreateRawTransactionOutput`- Serialization of data into protobuf data:

- `RawTransaction - String`

Example

```
client.createRawTransaction(input);
```

ExecuteTransaction

Call a read-only method on a contract.

Web API path

`/api/blockChain/executeTransaction`

POST

Parameters

`ExecuteTransactionDto` - Serialization of data into protobuf data:

- `RawTransaction - String`

Returns

`String`

Example

```
client.executeTransaction(input);
```

ExecuteRawTransaction

Call a read-only method on a contract.

Web API path

`/api/blockChain/executeRawTransaction`

POST

Parameters

`ExecuteRawTransactionDto` - Serialization of data into protobuf data:

- `RawTransaction - String`

- `Signature - String`

Returns

`String`

Example

```
client.executeRawTransaction(input);
```

GetPeers

Get peer info about the connected network nodes.

Web API path

`/api/net/peers`

Parameters

1. `withMetrics - boolean`

Returns

`List<PeerDto>`

- `IpAddress - String`
- `ProtocolVersion - int`
- `ConnectionTime - long`
- `ConnectionStatus - String`
- `Inbound - boolean`
- `BufferedTransactionsCount - int`
- `BufferedBlocksCount - int`
- `BufferedAnnouncementsCount - int`
- `RequestMetrics - List<RequestMetric>`
  - `RoundTripTime - long`
  - `MethodName - String`
  - `Info - String`
  - `RequestTime - String`

Example

```
client.getPeers(false);
```

AddPeer

Attempts to add a node to the connected network nodes.

Web API path

`/api/net/peer`

POST

Parameters

`AddPeerInput`

- `Address - String`

Returns

`boolean`

Example

```
client.addPeer("127.0.0.1:7001");
```

RemovePeer

Attempts to remove a node from the connected network nodes.

Web API path

`/api/net/peer`

DELETE

Parameters

1. `address - String`

Returns

`boolean`

Example

```
client.removePeer("127.0.0.1:7001");
```

GetNetworkInfo

Get the network information of the node.

Web API path

`/api/net/networkInfo`

Parameters

Empty

Returns

`NetworkInfoOutput`

- `Version - String`
- `ProtocolVersion - int`
- `Connections - int`

Example

```
client.getNetworkInfo();
```

18.4.5 AElf Client

IsConnected

Verify whether this sdk successfully connects the chain.

Parameters

Empty

Returns

`boolean`

Example

```
client.isConnected();
```

GetGenesisContractAddress

Get the address of genesis contract.

Parameters

Empty

Returns

`String`

Example

```
client.getGenesisContractAddress();
```

GetContractAddressByName

Get address of a contract by given contractNameHash.

Parameters

1. `privateKey - String`
2. `contractNameHash - byte[]`

Returns

`String`

Example

```
client.getContractAddressByName(privateKey, contractNameHash);
```

GenerateTransaction

Build a transaction from the input parameters.

Parameters

1. `from - String`

2. `to - String`
3. `methodName - String`
4. `input - byte[]`

Returns

`Transaction`

Example

```
client.generateTransaction(from, to, methodName, input);
```

GetFormattedAddress

Convert the Address to the displayed stringsymbol_base58-string_base58-String-chain-id.

Parameters

1. `privateKey - String`
2. `address - String`

Returns

`String`

Example

```
client.getFormattedAddress(privateKey, address);
```

SignTransaction

Sign a transaction using private key.

Parameters

1. `privateKeyHex - String`
2. `transaction - Transaction`

Returns

`String`

Example

```
client.signTransaction(privateKeyHex, transaction);
```

GetAddressFromPubKey

Get the account address through the public key.

Parameters

1. `pubKey - String`

Returns

`String`

Example

```
client.getAddressFromPubKey(pubKey);
```

GetAddressFromPrivateKey

Get the account address through the private key.

Parameters

1. `privateKey - String`

Returns

`String`

Example

```
client.getAddressFromPrivateKey(privateKey);
```

GenerateKeyPairInfo

Generate a new account key pair.

Parameters

Empty

Returns

`KeyPairInfo`

- `PrivateKey - String`
- `PublicKey - String`
- `Address - String`

Example

```
client.generateKeyPairInfo();
```

18.4.6 Supports

- JDK1.8+
- Log4j2.6.2

18.5 aelf-sdk.php - AELF PHP API

18.5.1 Introduction

aelf-sdk.php for aelf is like web.js for ethereum.

aelf-sdk.php is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.php, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.php)

18.5.2 Adding AElf php SDK

In order to install this library via composer run the following command in the console:

```
$ composer require aelf/aelf-sdk dev-dev
```

composer require curl/curl

If you directly clone the sdk You must install composer and execute it in the root directory

```
"aelf/aelf-sdk": "dev-dev"
```

18.5.3 Examples

You can also see full examples in `./test`;

1.Create instance

Create a new instance of AElf, connect to an AELF chain node. Using this instance, you can call the APIs on AElf.

```
require_once 'vendor/autoload.php';
use AElf\AElf;
$url = '127.0.0.1:8000';
$aelf = new AElf($url);
```

2.Get a system contract address

Get a system contract address, take `AElf.ContractNames.Token` as an example

```
require_once 'vendor/autoload.php';
use AElf\AElf;
$url = '127.0.0.1:8000';
$aelf = new AElf($url);

$privateKey = 'cd86ab6347d8e52bbbe8532141fc59ce596268143a308d1d40fedf385528b458';
$bytes = new Hash();
$bytes->setValue(hex2bin(hash('sha256', 'AElf.ContractNames.Token')));
$contractAddress = $aelf->GetContractAddressByName($privateKey, $bytes);
```

3.Send a transaction

Get the contract address, and then send the transaction.

```
require_once 'vendor/autoload.php';
use AElf\AElf;
$url = '127.0.0.1:8000';
// create a new instance of AElf
$aelf = new AElf($url);

// private key
$privateKey = 'cd86ab6347d8e52bbbe8532141fc59ce596268143a308d1d40fedf385528b458';

$aelfEcdsa = new BitcoinECDSA();
$aelfEcdsa->setPrivateKey($privateKey);
$publicKey = $aelfEcdsa->getUncompressedPubKey();
$address = $aelfEcdsa->hash256(hex2bin($publicKey));
$address = $address . substr($aelfEcdsa->hash256(hex2bin($address)), 0, 8);
// sender address
$base58Address = $aelfEcdsa->base58_encode($address);

// transaction input
$params = new Hash();
$params->setValue(hex2bin(hash('sha256', 'AElf.ContractNames.Vote')));

// transaction method name
$methodName = "GetContractAddressByName";

// transaction contract address
$toAddress = $aelf->getGenesisContractAddress();

// generate a transaction
$transactionObj = aelf->generateTransaction($base58Address, $toAddress, $methodName,
↪$params);

//signature
$signature = $aelf->signTransaction($privateKey, $transactionObj);
$transactionObj->setSignature(hex2bin($signature));

// obj Dto
$executeTransactionDtoObj = ['RawTransaction' => bin2hex($transaction->
↪serializeToString())];

$result = $aelf->sendTransaction($executeTransactionDtoObj);
print_r($result);
```

18.5.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

The usage of these methods is based on the AElf instance, so if you don't have one please create it:

```
require_once 'vendor/autoload.php';
use AElf\AElf;
$url = '127.0.0.1:8000';
// create a new instance of AElf
$aelf = new AElf($url);
```

1.getChainStatus

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`Array`

- `ChainId - String`
- `Branches - Array`
- `NotLinkedBlocks - Array`
- `LongestChainHeight - Integer`
- `LongestChainHash - String`
- `GenesisBlockHash - String`
- `GenesisContractAddress - String`
- `LastIrreversibleBlockHash - String`
- `LastIrreversibleBlockHeight - Integer`
- `BestChainHash - String`
- `BestChainHeight - Integer`

Example

```
// create a new instance of AElf
$aelf = new AElf($url);

$chainStatus = $aelf->getChainStatus();
print_r($chainStatus);
```

2.getBlockHeight

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`Integer`

Example

```
$aelf = new AElf($url);

$height = $aelfClient->GetBlockHeight();
print($height);
```

3.getBlock

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `block_hash - String`
2. `include_transactions - Boolean :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`Array`

- `BlockHash - String`
- `Header - Array`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - List`
  - `Height - Integer`
  - `Time - String`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - Array`
  - `TransactionsCount - Integer`
  - `Transactions - Array`
    * `transactionId - String`

Example

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
$block2 = $aelf->getBlockByHash($block['BlockHash'], false);
print_r($block2);
```

4.getBlockByHeight

Web API path

`/api/blockChain/blockByHeight`

Get block information by block height.

Parameters

1. `block_height - Number`
2. `include_transactions - Boolean :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`Array`

- `BlockHash - String`
- `Header - Array`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - List`
  - `Height - Integer`
  - `Time - String`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`
- `Body - Array`
  - `TransactionsCount - Integer`
  - `Transactions - Array`
    * `transactionId - String`

Example

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
print_r($block);
```

5.getTransactionResult

Get the result of a transaction

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - String`

Returns

`Object`

- `TransactionId - String`
- `Status - String`
- `Logs - Array`
  - `Address - String`
  - `Name - String`
  - `Indexed - Array`
  - `NonIndexed - String`
- `Bloom - String`
- `BlockNumber - Integer`
- `Transaction - Array`
  - `From - String`
  - `To - String`
  - `RefBlockNumber - Integer`
  - `RefBlockPrefix - String`
  - `MethodName - String`
  - `Params - json`
  - `Signature - String`
- `ReadableReturnValue - String`
- `Error - String`

Example

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
$transactionResult = $aelf->getTransactionResult($block['Body']['Transactions'][0]);
print_r('# get_transaction_result');
print_r($transactionResult);
```

6.getTransactionResults

Get multiple transaction results in a block

Web API path

`/api/blockChain/transactionResults`

Parameters

1. `blockHash - String`

2. `offset - Number`
3. `limit - Number`

Returns

`List` - The array of method descriptions:

- the transaction result object

Example

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
$transactionResults = $aelf->getTransactionResults($block['Body']);
print_r($transactionResults);
```

7.getTransactionPoolStatus

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Example

```
$aelf = new AElf($url);

$status = $aelf->getTransactionPoolStatus();
print_r($status);
```

8.sendTransaction

Broadcast a transaction

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`transaction - String` - Serialization of data into String

Example

```
$aelf = new AElf($url);

$params = new Hash();
$params->setValue(hex2bin(hash('sha256', 'AElf.ContractNames.Vote')));
$transaction = buildTransaction($aelf->getGenesisContractAddress(),
↪'GetContractAddressByName', $params);
$executeTransactionDtoObj = ['RawTransaction' => bin2hex($transaction->
↪serializeToString())];
$result = $aelf->sendTransaction($executeTransactionDtoObj);
print_r($result);
```

9.sendTransactions

Broadcast multiple transactions

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`transactions - String` - Serialization of data into String

Example

```
$aelf = new AElf($url);

$paramsList = [$params1, $params2];
$rawTransactionsList = [];
foreach ($paramsList as $param) {
    $transactionObj = buildTransaction($toAddress, $methodName, $param);
    $rawTransactions = bin2hex($transactionObj->serializeToString());
    array_push($rawTransactionsList, $rawTransactions);
}
$sendTransactionsInputs = ['RawTransactions' => implode(',', $rawTransactionsList)];
$listString = $this->aelf->sendTransactions($sendTransactionsInputs);
print_r($listString);
```

10.getPeers

Get peer info about the connected network nodes

Web API path

`/api/net/peers`

Example

```
$aelf = new AElf($url);

print_r($aelf->getPeers(true));
```

11.addPeer

Attempts to add a node to the connected network nodes

Web API path

`/api/net/peer`

POST

Parameters

`peer_address - String` - peer's endpoint

Example

```
$aelf = new AElf($url);

$aelf->addPeer($url);
```

12.removePeer

Attempts to remove a node from the connected network nodes

Web API path

`/api/net/peer?address=`

POST

Parameters

`peer_address - String` - peer's endpoint

Example

```
$aelf = new AElf($url);

$aelf->removePeer($url);
```

13.createRawTransaction

create a raw transaction

Web API path

`/api/blockchain/rawTransaction`

POST

Parameters

1. `transaction - Array`

Returns

`Array`

- `RawTransaction - hex string bytes generated by transaction information`

Example

```
$aelf = new AElf($url);

$status = $aelf->getChainStatus();
$params = base64_encode(hex2bin(hash('sha256', 'AElf.ContractNames.Consensus')));
$param = array('value' => $params);
$transaction = [
    "from" => $aelf->getAddressFromPrivateKey($privateKey),
    "to" => $aelf->getGenesisContractAddress(),
    "refBlockNumber" => $status['BestChainHeight'],
    "refBlockHash" => $status['BestChainHash'],
    "methodName" => "GetContractAddressByName",
    "params" => json_encode($param)
];
```

(continues on next page)

(continued from previous page)

```
$rawTransaction = $aelf->createRawTransaction($transaction);
print_r($rawTransaction);
```

14.sendRawTransaction

send raw transactions

Web API path

`/api/blockchain/sendRawTransaction`

Parameters

1. `Transaction - raw transaction`
2. `Signature - signature`
3. `ReturnTransaction - indicates whether to return transaction`

Example

```
$aelf = new AElf($url);

$rawTransaction = $aelf->createRawTransaction($transaction);
$transactionId = hash('sha256', hex2bin($rawTransaction['RawTransaction']));
$sign = $aelf->getSignatureWithPrivateKey($privateKey, $transactionId);
$transaction = array('Transaction' => $rawTransaction['RawTransaction'], 'signature'␣
↪=> $sign, 'returnTransaction' => true);
$execute = $aelf->sendRawTransaction($transaction);
print_r($execute);
```

15.executeRawTransaction

execute raw transactions

Web API path

`/api/blockchain/executeRawTransaction`

Post

Parameters

1. `RawTransaction - raw transaction`
2. `Signature - signature`

Example

```
$aelf = new AElf($url);

$rawTransaction = $aelf->createRawTransaction($transaction);
$transactionId = hash('sha256', hex2bin($rawTransaction['RawTransaction']));
$sign = $aelf->getSignatureWithPrivateKey($privateKey, $transactionId);
$transaction = array('RawTransaction' => $rawTransaction['RawTransaction'], 'signature
↪' => $sign);
$execute = $aelf->executeRawTransaction($transaction);
print_r($execute);
```

16.getMerklePathByTransactionId

get merkle path

Web API path

`/api/blockchain/merklePathByTransactionId?transactionId=`

Parameters

1. `transactionId - String`

Example

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
$merklePath = $aelf->getMerklePathByTransactionId($block['Body']['Transactions'][0]);
```

17.getNetworkInfo

get network information

Web API path

`/api/net/networkInfo`

Example

```
$aelf = new AElf($url);

print_r($aelf->getNetworkInfo());
```

18.getContractFileDescriptorSet

get contract file descriptor set

Web API path

`/api/blockChain/contractFileDescriptorSet`

Example

```
$aelf = new AElf($url);

$blockDto = $aelf->getBlockByHeight($blockHeight, false);
$transactionResultDtoList = $aelf->getTransactionResults($blockDto['BlockHash'], 0,␣
↪10);
foreach ($transactionResultDtoList as $v) {
  $request = $aelf->getContractFileDescriptorSet($v['Transaction']['To']);
  print_r($request);
}
```

19.getTaskQueueStatus

get task queue status

Web API path

`/api/blockChain/taskQueueStatus`

Example

```
$aelf = new AElf($url);

$taskQueueStatus = $aelf->getTaskQueueStatus();
print_r($taskQueueStatus);
```

20.executeTransaction

execute transaction

Web API path

Post

`/api/blockChain/executeTransaction`

Example

```
$aelf = new AElf($url);

$methodName = "GetNativeTokenInfo";
$bytes = new Hash();
$bytes->setValue(hex2bin(hash('sha256', 'AElf.ContractNames.Token')));
$toAddress = $aelf->GetContractAddressByName($privateKey, $bytes);
$param = new Hash();
$param->setValue('');
$transaction = $aelf->generateTransaction($fromAddress, $toAddress, $methodName,
↪$param);
$signature = $aelf->signTransaction($privateKey, $transaction);
$transaction->setSignature(hex2bin($signature));
$executeTransactionDtoObj = ['RawTransaction' => bin2hex($transaction->
↪serializeToString())];
$response = $aelf->executeTransaction($executeTransactionDtoObj);
$tokenInfo = new TokenInfo();
$tokenInfo->mergeFromString(hex2bin($response));
```

18.5.5 Other Tool Kit

AElf supply some APIs to simplify developing.

1.getChainId

get chain id

```
$aelf = new AElf($url);

$chainId = $aelf->getChainId();
print_r($chainId);
```

2.generateTransaction

generate a transaction object

```
$aelf = new AElf($url);

$param = new Hash();
$param->setValue('');
$transaction = $aelf->generateTransaction($fromAddress, $toAddress, $methodName,
↪$param);
```

3.signTransaction

sign a transaction

```
$aelf = new AElf($url);

$transaction = $aelf->generateTransaction($fromAddress, $toAddress, $methodName,
↪$param);
$signature = $aelf->signTransaction($privateKey, $transaction);
```

4.getGenesisContractAddress

get the genesis contract's address

```
$aelf = new AElf($url);

$genesisContractAddress = $aelf->getGenesisContractAddress();
print_r($genesisContractAddress);
```

4.getAddressFromPubKey

calculate the account address accoriding to the public key

```
$aelf = new AElf($url);

$pubKeyAddress = $aelf->getAddressFromPubKey(
↪'04166cf4be901dee1c21f3d97b9e4818f229bec72a5ecd56b5c4d6ce7abfc3c87e25c36fd279db721acf4258fb489b4a4
↪');
print_r($pubKeyAddress);
```

5.getFormattedAddress

convert the Address to the displayed stringsymbol_base58-string_base58-string-chain-id.

```
$aelf = new AElf($url);

$addressVal = $aelf->getFormattedAddress($privateKey, $base58Address);
print_r($addressVal);
```

6.generateKeyPairInfo

generate a new key pair using ECDSA

```
$aelf = new AElf($url);

$pairInfo = $aelf->generateKeyPairInfo();
print_r($pairInfo);
```

7.getContractAddressByName

get contract's address from its name

```
$aelf = new AElf($url);

$bytes = new Hash();
$bytes->setValue(hex2bin(hash('sha256', 'AElf.ContractNames.Token')));
$contractAddress = $aelf->GetContractAddressByName($privateKey, $bytes);
print_r($contractAddress);
```

8.getAddressFromPrivateKey

get address from a private key

```
$aelf = new AElf($url);

$address = $aelf->getAddressFromPrivateKey($privateKey);
print_r($address);
```

9.getSignatureWithPrivateKey

given a private key, get the signature

```
$aelf = new AElf($url);

$sign = $aelf->getSignatureWithPrivateKey($privateKey, $transactionId);
print_r($sign);
```

10.isConnected

check if it connects the chain

```
$aelf = new AElf($url);

$isConnected = $this->aelf->isConnected();
print_r($isConnected);
```

11.getTransactionFees

get the transaction fee from transaction result

```
$aelf = new AElf($url);

$block = $aelf->getBlockByHeight(1, true);
$transactionResult = $aelf->getTransactionResult($block['Body']['Transactions'][0]);
$transactionFees = $aelf->getTransactionFees($transactionResult);
print_r($transactionFees);
```

18.5.6 AElf.version

```
$aelf = new AElf($url);

$version = $aelf->version;
```

18.5.7 Requirements

- php

18.5.8 About contributing

Read out [contributing guide]

18.5.9 About Version

https://semver.org/

18.6 aelf-sdk.py - AELF Python API

18.6.1 Introduction

aelf-sdk.py for aelf is like web.js for ethereum.

aelf-sdk.py is a collection of libraries which allow you to interact with a local or remote aelf node, using a HTTP connection.

The following documentation will guide you through installing and running aelf-sdk.py, as well as providing a API reference documentation with examples.

If you need more information you can check out the repo : aelf-sdk.py

18.6.2 Adding aelf-sdk.js

First you need to get aelf-sdk package into your project. This can be done using the following methods:

pip: `pip install aelf-sdk`

After that you need to create a aelf instance by a node's URL.

```
    chain = AElf('http://127.0.0.1:8000')
```

18.6.3 Examples

You can also see full examples in `./test`;

1.Create instance

Create a new instance of AElf, connect to an AELF chain node. Using this instance, you can call the APIs on AElf.

```
    from aelf import AElf

    // create a new instance of AElf
    aelf = AElf('http://127.0.0.1:8000')
```

2.Get a system contract address

Get a system contract address, take `AElf.ContractNames.Token` as an example

```
from aelf import AElf

aelf = AElf('http://127.0.0.1:8000')
// get genesis contract address
genesis_contract_address = aelf.get_genesis_contract_address_string()

// get contract address
// in fact, get_system_contract_address call the method 'GetContractAddressByName' in␣
↪the genesis contract to get other contracts' address
multi_token_contract_address = aelf.get_system_contract_address('AElf.ContractNames.
↪Token')
```

3.Send a transaction

Get the contract address, and then send the transaction.

```
from aelf import AElf

url = 'http://127.0.0.1:8000'
// create a new instance of AElf
aelf = AElf(url)

// generate the private key
private_key_string = 'b344570eb80043d7c5ae9800c813b8842660898bf03cbd41e583b4e54af4e7fa
↪'
private_key = PrivateKey(bytes(bytearray.fromhex(private_key_string)))

// create input, the type is generated by protoc
cross_chain_transfer_input = CrossChainTransferInput()

// generate the transaction
transaction = aelf.create_transaction(to_address, method_name, params.
↪SerializeToString())

// sign the transaction by user's private key
aelf.sign_transaction(private_key, transaction)
```

(continues on next page)

(continued from previous page)

```
// execute the transaction
aelf.execute_transaction(transaction)
```

18.6.4 Web API

You can see how the Web Api of the node works in `{chainAddress}/swagger/index.html` *tip: for an example, my local address: 'http://127.0.0.1:1235/swagger/index.html'*

The usage of these methods is based on the AElf instance, so if you don't have one please create it:

```
from aelf import AElf

// create a new instance of AElf, change the URL if needed
aelf = AElf('http://127.0.0.1:8000')
```

1.get_chain_status

Get the current status of the block chain.

Web API path

`/api/blockChain/chainStatus`

Parameters

Empty

Returns

`json`

- `ChainId - String`
- `Branches - json`
- `NotLinkedBlocks - json`
- `LongestChainHeight - Number`
- `LongestChainHash - String`
- `GenesisBlockHash - String`
- `GenesisContractAddress - String`
- `LastIrreversibleBlockHash - String`
- `LastIrreversibleBlockHeight - Number`
- `BestChainHash - String`
- `BestChainHeight - Number`

Example

```
aelf = AElf(url)

chain_status = aelf.get_chain_status()
print('# get_chain_status', chain_status)
```

2.get_block_height

Get current best height of the chain.

Web API path

`/api/blockChain/blockHeight`

Parameters

Empty

Returns

`Number`

Example

```
aelf = AElf(url)

block_height = aelf.get_block_height()
print('# get_block_height', block_height)
```

3.get_block

Get block information by block hash.

Web API path

`/api/blockChain/block`

Parameters

1. `block_hash - String`
2. `include_transactions - Boolean :`

- `true` require transaction ids list in the block
- `false` Doesn't require transaction ids list in the block

Returns

`json`

- `BlockHash - String`
- `Header - json`
  - `PreviousBlockHash - String`
  - `MerkleTreeRootOfTransactions - String`
  - `MerkleTreeRootOfWorldState - String`
  - `Extra - List`
  - `Height - Number`
  - `Time - json`
  - `ChainId - String`
  - `Bloom - String`
  - `SignerPubkey - String`

- Body - json
  - TransactionsCount - Number
  - Transactions - List
    * transactionId - String

Example

```
aelf = AElf(url)

block = aelf.get_block(blockHash)
print('# get_block', block)
```

4.get_block_by_height

Web API path

/api/blockChain/blockByHeight

Get block information by block height.

Parameters

1. block_height - Number
2. include_transactions - Boolean :

- true require transaction ids list in the block
- false Doesn't require transaction ids list in the block

Returns

json

- BlockHash - String
- Header - json
  - PreviousBlockHash - String
  - MerkleTreeRootOfTransactions - String
  - MerkleTreeRootOfWorldState - String
  - Extra - List
  - Height - Number
  - Time - json
  - ChainId - String
  - Bloom - String
  - SignerPubkey - String
- Body - json
  - TransactionsCount - Number
  - Transactions - List
    * transactionId - String

Example

```
aelf = AElf(url)

block_by_height = aelf.get_block_by_height(12, false)
print('# get_block_by_height', block_by_height)
```

5.get_transaction_result

Get the result of a transaction

Web API path

`/api/blockChain/transactionResult`

Parameters

1. `transactionId - String`

Returns

`json`

- `TransactionId - String`
- `Status - String`
- `Logs - List`
  - `Address - String`
  - `Name - String`
  - `Indexed - List`
  - `NonIndexed - String`
- `Bloom - String`
- `BlockNumber - Number`
- `Transaction - List`
  - `From - String`
  - `To - String`
  - `RefBlockNumber - Number`
  - `RefBlockPrefix - String`
  - `MethodName - String`
  - `Params - json`
  - `Signature - String`
- `ReadableReturnValue - json`
- `Error - String`

Example

```
aelf = AElf(url)

transaction_result = aelf.get_transaction_result(transactionId)
print('# get_transaction_results', transaction_result)
```

6.get_transaction_results

Get multiple transaction results in a block

Web API path

`/api/blockChain/transactionResults`

Parameters

1. `blockHash - String`
2. `offset - Number`
3. `limit - Number`

Returns

`List` - The array of method descriptions:

- the transaction result object

Example

```
aelf = AElf(url)

transaction_results = aelf.get_transaction_results(blockHash, 0, 2)
print('# get_transaction_results', transaction_results)
```

7.get_transaction_pool_status

Get the transaction pool status.

Web API path

`/api/blockChain/transactionPoolStatus`

Example

```
aelf = AElf(url)

tx_pool_status = aelf.get_transaction_pool_status()
print('# get_transaction_pool_status', tx_pool_status)
```

8.send_transaction

Broadcast a transaction

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`transaction - String` - Serialization of data into String

Example

```
aelf = AElf(url)

current_height = aelf.get_block_height()
block = aelf.get_block_by_height(current_height, include_transactions=False)
transaction = Transaction()
transaction.to_address.CopyFrom(aelf.get_system_contract_address("AElf.ContractNames.
↪Consensus"))
transaction.ref_block_number = current_height
transaction.ref_block_prefix = bytes.fromhex(block['BlockHash'])[0:4]
transaction.method_name = 'GetCurrentMinerList'
transaction = aelf.sign_transaction(private_key, transaction)
result = aelf.send_transaction(transaction.SerializePartialToString().hex())
print('# send_transaction', result)
```

9.send_transactions

Broadcast multiple transactions

Web API path

`/api/blockChain/sendTransaction`

POST

Parameters

`transactions - String` - Serialization of data into String

Example

```
aelf = AElf(url)

current_height = aelf.get_block_height()
block = aelf.get_block_by_height(current_height, include_transactions=False)
transaction1 = Transaction().SerializePartialToString().hex()
transaction2 = Transaction().SerializePartialToString().hex()
result = aelf.send_transaction(transaction1 + ',' + transaction2)
print('# send_transactions', result)
```

10.get_peers

Get peer info about the connected network nodes

Web API path

`/api/net/peers`

Example

```
aelf = AElf(url)

peers = aelf.get_peers()
print('# get_peers', peers)
```

11.add_peer

Attempts to add a node to the connected network nodes

Web API path

`/api/net/peer`

POST

Parameters

`peer_address - String` - peer's endpoint

Example

```
aelf = AElf(url)

add_peer = aelf.add_peer(endpoint)
print('# add_peers', add_peer)
```

12.remove_peer

Attempts to remove a node from the connected network nodes

Web API path

`/api/net/peer?address=`

POST

Parameters

`peer_address - String` - peer's endpoint

Example

```
aelf = AElf(url)

remove_peer = aelf.remove_peer(address)
print('# remove_peer', remove_peer)
```

13.create_raw_transaction

create a raw transaction

Web API path

`/api/blockchain/rawTransaction`

POST

Parameters

1. `transaction - the json format transaction`

Returns

`json`

- `RawTransaction - hex string bytes generated by transaction information`

Example

```
aelf = AElf(url)

transaction = {
    "From": aelf.get_address_string_from_public_key(public_key),
    "To": aelf.get_system_contract_address_string("AElf.ContractNames.Consensus"),
    "RefBlockNumber": 0,
    "RefBlockHash": "b344570eb80043d7c5ae9800c813b8842660898bf03cbd41e583b4e54af4e7fa
↪",
    "MethodName": "GetCurrentMinerList",
    "Params": '{}'
}
raw_transaction = aelf.create_raw_transaction(transaction)
```

14.send_raw_transaction

send raw transactions

Web API path

`/api/blockchain/sendRawTransaction`

Parameters

1. `Transaction - raw transaction`
2. `Signature - signature`
3. `ReturnTransaction - indicates whether to return transaction`

Example

```
aelf = AElf(url)

raw_transaction = aelf.create_raw_transaction(transaction)
signature = private_key.sign_recoverable(bytes.fromhex(raw_transaction['RawTransaction
↪']))
transaction_2 = {
  "Transaction": raw_transaction['RawTransaction'],
  'Signature': signature.hex(),
  'ReturnTransaction': True
}
print('# send_raw_transaction', aelf.send_raw_transaction(transaction_2))
```

15.execute_raw_transaction

execute raw transactions

Web API path

`/api/blockchain/executeRawTransaction`

Post

Parameters

1. `RawTransaction - raw transaction`
2. `Signature - signature`

Example

```
aelf = AElf(url)

raw_transaction = aelf.create_raw_transaction(transaction)
signature = private_key.sign_recoverable(bytes.fromhex(raw_transaction['RawTransaction
↪']))
transaction_1 = {
  "RawTransaction": raw_transaction['RawTransaction'],
  "Signature": signature.hex()
}
print('# execute_raw_transaction', aelf.execute_raw_transaction(transaction_1))
```

16.get_merkle_path

get merkle path

Web API path

`/api/blockchain/merklePathByTransactionId?transactionId=`

Parameters

1. `transactionId - String`

Example

```
aelf = AElf(url)

transaction_results = aelf.get_transaction_results(transactionId)
print('# get_transaction_results', transaction_results)
```

17.get_network_info

get network information

Web API path

`/api/net/networkInfo`

Example

```
aelf = AElf(url)

print('# get_network_info', aelf.get_network_info())
```

18.6.5 AElf.client

Use the api to see detailed results

1.get_genesis_contract_address_string

Returns

`String`: zero contract address

Example

```
aelf = AElf(url)

genesis_contract_address = aelf.get_genesis_contract_address_string()
```

2.get_system_contract_address

Parameters

1. `contract_name - String` : system Contract's name

Returns

`Address`: system Contract's address

Example

```
aelf = AElf(url)

multi_token_contract_address = aelf.get_system_contract_address('AElf.ContractNames.
↪Token')
```

3.get_system_contract_address_string

Parameters

1. `contract_name - String` : system Contract's name

Returns

`String`: system Contract's address

Example

```
aelf = AElf(url)

multi_token_contract_address_string = aelf.get_system_contract_address_string('AElf.
↪ContractNames.Token')
```

4.create_transaction

create a transaction

Parameters

1. `to_address - Address or String` : target contract's address
2. `method_name - String` : method name
3. `params - String` : serilize paramters into String

Example

```
aelf = AElf(url)

params = Hash()
params.value = hashlib.sha256(contract_name.encode('utf8')).digest()
transaction = self.create_transaction(genesisContractAddress,
↪'GetContractAddressByName', params.SerializeToString())
```

5.sign_transaction

sign transaction with user's private key

Parameters

1. `private_key - String` : user's private key
2. `transaction - Transaction` : transaction

Example_

```
aelf = AElf(url)

to_address_string = aelf.get_genesis_contract_address_string()
params = Hash()
params.value = hashlib.sha256(contract_name.encode('utf8')).digest()
transaction = aelf.create_transaction(to_address_string, 'GetContractAddressByName',
↪params.SerializeToString())
transaction = aelf.sign_transaction(private_key, transaction)
```

6.get_address_from_public_key

generate address from public key

Parameters

1. `public_key - bytes` : user's pubilc key

Returns

`Address`

Example_

```
aelf = AElf(url)

address = aelf.get_address_from_public_key(public_key)
```

7.get_address_string_from_public_key

generate address string from public key

Parameters

1. `public_key - bytes` : user's pubilc key

Returns

`String`

Example_

```
aelf = AElf(url)

address = aelf.get_address_string_from_public_key(public_key)
```

8.get_chain_id

get chain id

Returns

`Number`

Example_

```
aelf = AElf(url)

chain_id = aelf.get_chain_id()
print('# get_chain_id', chain_id)
```

9.get_formatted_address

get formatted address

Parameters

1. `address Address` : address

Returns

`String`

Example_

```
aelf = AElf(url)

address = aelf.chain.get_system_contract_address("AElf.ContractNames.Consensus")
formatted_address = aelf.get_formatted_address(address)
print('formatted address', formatted_address)
```

10.is_connected

check whether to connect the node

Example_

```
aelf = AElf(url)

is_connected = aelf.is_connected()
```

18.6.6 Tookkits.py

AElfToolkit Encapsulate AElf and user's private key. It simplifies the procedures of sending some transactions. You can find it in src/aelf/toolkits.py.

Create a toolKit

Create a toolKit with `AElfToolkit`.

```
from aelf import AElfToolkit

// generate the private key
private_key_string = 'b344570eb80043d7c5ae9800c813b8842660898bf03cbd41e583b4e54af4e7fa
↪'
private_key = PrivateKey(bytes(bytearray.fromhex(private_key_string)))
// create a toolKit
toolkit = AElfToolkit('http://127.0.0.1:8000', private_key)
```

Send a transaction

Send a CrossChainTransfer transaction

```
from aelf import AElfToolkit

// generate the private key
private_key_string = 'b344570eb80043d7c5ae9800c813b8842660898bf03cbd41e583b4e54af4e7fa
↪'
private_key = PrivateKey(bytes(bytearray.fromhex(private_key_string)))

// create input, the type is generated by protoc
cross_chain_transfer_input = CrossChainTransferInput()

// AElfToolkit simplifies this transcation execution.
// create a toolKit
toolkit = AElfToolkit('http://127.0.0.1:8000', private_key)
toolkit.cross_chain_transfer(to_address_string, symbol, amount, memo, to_chain_id)
```

18.6.7 Requirements

- python
- docker

18.6.8 Support

node

18.6.9 About contributing

Read out [contributing guide]

18.6.10 About Version

https://semver.org/

CHAPTER 19

C# reference

19.1 AElf.Sdk.CSharp

19.1.1 Contents

BoolState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around boolean values for use in smart contract state.

BytesState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around byte arrays for use in smart contract state.

CSharpSmartContractContext `type`

Namespace

AElf.Sdk.CSharp

Summary

Represents the transaction execution context in a smart contract. An instance of this class is present in the base class for smart contracts (Context property). It provides access to properties and methods useful for implementing the logic in smart contracts.

ChainId `property`

Summary

The chain id of the chain on which the contract is currently running.

CurrentBlockTime `property`

Summary

The time included in the current blocks header.

CurrentHeight `property`

Summary

The height of the block that contains the transaction currently executing.

Origin `property`

Summary

The address of the sender (signer) of the transaction being executed. It's type is an AElf address. It corresponds to the From field of the transaction. This value never changes, even for nested inline calls. This means that when you access this property in your contract, it's value will be the entity that created the transaction (user or smart contract through an inline call).

PreviousBlockHash `property`

Summary

The hash of the block that precedes the current in the blockchain structure.

Self `property`

Summary

The address of the contract currently being executed. This changes for every transaction and inline transaction.

Sender `property`

Summary

The Sender of the transaction that is executing.

StateProvider `property`

Summary

Provides access to the underlying state provider.

TransactionId `property`

Summary

The ID of the transaction that's currently executing.

Variables `property`

Summary

Provides access to variable of the bridge.

Transaction `property`

Summary

Including some transaction info.

Call(fromAddress,toAddress,methodName,args) `method`

Summary

Calls a method on another contract.

Returns

The result of the call.

Parameters

Name	Type	Description
fromAddress	AElf.Types.Address	The address to use as sender.
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to call.
args	Google.Protobuf.ByteString	The input arguments for calling that method. This is usually generated from the protobuf
definition of the input type		

Generic Types

Name	Description
T	The type of the return message.

ConvertHashToInt64(hash,start,end) `method`

Summary

Converts the input hash to a 64-bit signed integer.

Returns

The 64-bit signed integer.

Parameters

Name	Type	Description
hash	AElf.Types.Hash	The hash.
start	System.Int64	The inclusive lower bound of the number returned.
end	System.Int64	The exclusive upper bound of the number returned. endValue must be greater than or equal to startValue.

Exceptions

Name	Description
System.ArgumentException	startValue is less than 0 or greater than endValue.

ConvertVirtualAddressToContractAddress(virtualAddress) `method`

Summary

Converts a virtual address to a contract address.

Returns

The converted address.

Parameters

Name	Type	Description
virtualAddress	AElf.Types.Hash	The virtual address that want to convert.

ConvertVirtualAddressToContractAddress(virtualAddress,contractAddress) `method`

Summary

Converts a virtual address to a contract address with the contract address.

Returns

The converted address.

Parameters

Name	Type	Description
virtualAddress	AElf.Types.Hash	The virtual address that want to convert.
contractAddress	AElf.Types.Address	The contract address.

ConvertVirtualAddressToContractAddressWithContractHashName(virtualAddress) `method`

Summary

Converts a virtual address to a contract address with the current contract hash name.

Returns

The converted address.

Parameters

Name	Type	Description
virtualAddress	AElf.Types.Hash	The virtual address that want to convert.

ConvertVirtualAddressToContractAddressWithContractHashName(

virtualAddress,contractAddress) `method`

Summary

Converts a virtual address to a contract address with the contract hash name.

Returns

Parameters

Name	Type	Description
virtualAddress	AElf.Types.Hash	The virtual address that want to convert.
contractAddress	AElf.Types.Address	The contract address.

DeployContract(address,registration,name) `method`

Summary

Deploy a new smart contract (only the genesis contract can call it).

Parameters

Name	Type	Description
address	AElf.Types.Address	The address of new smart contract.
registration	AElf.Types.SmartContractRegistration	The registration of the new smart contract.
name	AElf.Types.Hash	The hash value of the smart contract name.

FireLogEvent(logEvent) `method`

Summary

This method is used to produce logs that can be found in the transaction result after execution.

Parameters

Name	Type	Description
logEvent	AElf.Types.LogEvent	The event to fire.

GenerateId(contractAddress,bytes) `method`

Summary

Generate a hash type id based on the contract address and the bytes.

Returns

The generated hash type id.

Parameters

Name	Type	Description
contractAddress	AElf.Types.Address	The contract address on which the id generation is based.
bytes	System.Collections.Generic.IEnumerable{System.Byte}	The bytes on which the id generation is based.

GetContractAddressByName(hash) `method`

Summary

It's sometimes useful to get the address of a system contract. The input is a hash of the system contracts name. These hashes are easily accessible through the constants in the SmartContractConstants.cs file of the C# SDK.

Returns

The address of the system contract.

Parameters

Name	Type	Description
hash	AElf.Types.Hash	The hash of the name.

GetPreviousBlockTransactions() `method`

Summary

Returns the transaction included in the previous block (previous to the one currently executing).

Returns

A list of transaction.

Parameters

This method has no parameters.

GetRandomHash(fromHash) `method`

Summary

Gets a random hash based on the input hash.

Returns

Random hash.

Parameters

Name	Type	Description
fromHash	AElf.Types.Hash	Hash.

GetSystemContractNameToAddressMapping() `method`

Summary

Get the mapping that associates the system contract addresses and their name's hash.

Returns

The addresses with their hashes.

Parameters

This method has no parameters.

GetZeroSmartContractAddress() `method`

Summary

This method returns the address of the Genesis contract (smart contract zero) of the current chain.

Returns

The address of the genesis contract.

Parameters

This method has no parameters.

GetZeroSmartContractAddress(chainId) `method`

Summary

This method returns the address of the Genesis contract (smart contract zero) of the specified chain.

Returns

The address of the genesis contract, for the given chain.

Parameters

Name	Type	Description
chainId	System.Int32	The chain's ID.

LogDebug(func) `method`

Summary

Application logging - when writing a contract it is useful to be able to log some elements in the applications log file to simplify development. Note that these logs are only visible when the node executing the transaction is build in debug mode.

Parameters

Name	Type	Description
func	System.Func {System.String}	The logic that will be executed for logging purposes.

RecoverPublicKey() `method`

Summary

Recovers the public key of the transaction Sender.

Returns

A byte array representing the public key.

Parameters

This method has no parameters.

SendInline(toAddress,methodName,args) `method`

Summary

Sends an inline transaction to another contract.

Parameters

Name	Type	Description
toAddress	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
args	Google.Protobuf .ByteString	The input arguments for calling that method. This is usually generated from the protobuf
definition of the input type.		

SendVirtualInline(fromVirtualAddress,toAddress,methodName,args) `method`

Summary

Sends a virtual inline transaction to another contract.

Parameters

Name	Type	Description
fromVirtualAddress	AElf.Types.Hash	The virtual address to use as sender.
toAddress	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
args	Google.Protobuf .ByteString	The input arguments for calling that method. This is usually generated from the protobuf
definition of the input type.		

SendVirtualInlineBySystemContract(fromVirtualAddress,toAddress,

methodName,args) `method`

Summary

Like SendVirtualInline but the virtual address us a system smart contract.

Parameters

Name	Type	Description
fromVirtualAddress	AElf.Types.Hash	Sends a virtual inline transaction to another contract. This method is only available to system smart contract.
toAddress	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
args	Google.Protobuf .ByteString	The input arguments for calling that method. This is usually generated from the protobuf
definition of the input type.		

UpdateContract(address,registration,name) `method`

Summary

Update a smart contract (only the genesis contract can call it).

Parameters

Name	Type	Description
address	AElf.Types.Address	The address of smart contract to update.
registration	AElf.Types.SmartContractRegistration	The registration of the smart contract to update.
name	AElf.Types.Hash <#T-AElf-Types-Hash>	The hash value of the smart contract name to update.

ValidateStateSize(obj) `method`

Summary

Verify that the state size is within the valid value.

Returns

The state.

Parameters

Name	Type	Description
obj	System.Object	The state.

Exceptions

Name	Description
AElf.Kernel.SmartContract.StateOverSizeException	The state size exceeds the limit.

VerifySignature(tx) `method`

Summary

Returns whether or not the given transaction is well formed and the signature is correct.

Returns

The verification results.

Parameters

Name	Type	Description
tx	AElf.Types.Transaction	The transaction to verify.

CSharpSmartContract `type`

Namespace

AElf.Sdk.CSharp

Summary

This class represents a base class for contracts written in the C# language. The generated code from the protobuf definitions will inherit from this class.

Generic Types

Name	Description
TContractState	

Context `property`

Summary

Represents the transaction execution context in a smart contract. It provides access inside the contract to properties and methods useful for implementing the smart contracts action logic.

State `property`

Summary

Provides access to the State class instance. TContractState is the type of the state class defined by the contract author.

ContractState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Base class for the state class in smart contracts.

Int32State `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around 32-bit integer values for use in smart contract state.

Int64State `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around 64-bit integer values for use in smart contract state.

MappedState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Key-value pair data structure used for representing state in contracts.

Generic Types

Name	Description
TKey	The type of the key.
TEntity	The type of the value.

SingletonState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Represents single values of a given type, for use in smart contract state.

SmartContractBridgeContextExtensions `type`

Namespace

AElf.Sdk.CSharp

Summary

Extension methods that help with the interactions with the smart contract execution context.

Call(context,address,methodName,message) `method`

Summary

Calls a method on another contract.

Returns

The return value of the call.

Parameters

Name	Type	Description
context	AElf.Kernel.SmartContract. ISmartContractBridgeContext	The virtual address of the system. contract to use as sender.
address	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to call.
message	Google.Protobuf.ByteString	The input arguments for calling that method. This is usually generated from the protobuf
definition of the input type.		

Generic Types

Name	Description
T	The return type of the call.

Call(context,address,methodName,message) `method`

Summary

Calls a method on another contract.

Returns

The result of the call.

Parameters

Name	Type	Description
context	*AElf.Sdk.CSharp.CSharpSmartContractContext*	An instance of ISmartContractBridgeContext
address	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to call.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

Generic Types

Name	Description
T	The type of the return message.

Call(context,fromAddress,toAddress,methodName,message) `method`

Summary

Calls a method on another contract.

Returns

The result of the call.

Parameters

Name	Type	Description
context	*AElf.Sdk.CSharp.CSharpSmartContractContext*	An instance of ISmartContractBridgeContext
fromAddress	AElf.Types. Address	The address to use as sender.
toAddressvv	AElf.Types. Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to call.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

Generic Types

Name	Description
T	The type of the return message.

Call(context,address,methodName,message) `method`

Summary

Calls a method on another contract.

Returns

The result of the call.

Parameters

Name	Type	Description
context	*AElf.Sdk.CSharp.CSharpSmartContractContext*	An instance of ISmartContractBridgeContext
address	AElf.Types. Address	The address to use as sender.
method-Name	System.String	The name of method you want to call.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

Generic Types

Name	Description
T	The type of the return message.

ConvertToByteString(message) `method`

Summary

Serializes a protobuf message to a protobuf ByteString.

Returns

ByteString.Empty if the message is null

Parameters

Name	Type	Description
message	Google.Protobuf.IMessage	The message to serialize.

ConvertVirtualAddressToContractAddress(this,virtualAddress) `method`

Summary

Converts a virtual address to a contract address.

Returns

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
virtualAddress	AElf.Types.Hash Address	The virtual address that want to convert.

ConvertVirtualAddressToContractAddressWithContractHashName(this, virtualAddress) `method`

Summary

Converts a virtual address to a contract address with the currently running contract address.

Returns

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
virtualAddress	AElf.Types.Hash Address	The virtual address that want to convert.

Fire(context,eventData) `method`

Summary

Logs an event during the execution of a transaction. The event type is defined in the AElf.CSharp.core project.

Parameters

Name	Type	Description
context	*AElf.Sdk.CSharp.CSharpSmartContractContext*	An instance of ISmartContractBridgeContext
eventData		The event to log.

Generic Types

Name	Description
T	The type of the event.

GenerateId(this,bytes) `method`

Summary

Generate a hash type id based on the currently running contract address and the bytes.

Returns

The generated hash type id.

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
bytes	System.Collections.Generic .IEnumerable{System.Byte}	The bytes on which the id generation is based.

GenerateId(this,token) `method`

Summary

Generate a hash type id based on the currently running contract address and the token.

Returns

The generated hash type id.

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
token	System.String	The token on which the id generation is based.

GenerateId(this,token) `method`

Summary

Generate a hash type id based on the currently running contract address and the hash type token.

Returns

The generated hash type id.

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridge-Context	An instance of ISmartContractBridgeContext
token	AElf.Types.Hash	The hash type token on which the id generation is based.

GenerateId(this) `method`

Summary

Generate a hash type id based on the currently running contract address.

Returns

The generated hash type id.

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContractBridgeCon-text	An instance of ISmartContractBridgeCon-text

GenerateId(this,address,token) `method`

Summary

Generate a hash type id based on the address and the bytes.

Returns

The generated hash type id.

Parameters

Name	Type	Description
this	AElf.Kernel.SmartContract. ISmartContract-BridgeContext	An instance of ISmartContractBridgeContext
ad-dress	AElf.Types.Address	The address on which the id generation is based.
token	AElf.Types.Hash	The hash type token on which the id generation is based.

SendInline(context,toAddress,methodName,message) `method`

Summary

Sends an inline transaction to another contract.

Parameters

Name	Type	Description
context	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

SendInline(context,toAddress,methodName,message) `method`

Summary

Sends a virtual inline transaction to another contract.

Parameters

Name	Type	Description
context	AElf.Kernel.SmartContract. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

SendVirtualInline(context,fromVirtualAddress,toAddress,methodName, message) `method`

Summary

Sends a virtual inline transaction to another contract.

Parameters

Name	Type	Description
context	AElf.Kernel.SmartContract. ISmartContract-BridgeContext	An instance of ISmartContractBridgeContext
fromVirtualAd-dress	AElf.Types.Hash	The virtual address to use as sender.
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

SendVirtualInline(context,fromVirtualAddress,toAddress,methodName, message) `method`

Summary

Sends a virtual inline transaction to another contract.

Parameters

Name	Type	Description
context	AElf.Kernel.SmartContract. ISmartContract-BridgeContext	An instance of ISmartContractBridgeContext
fromVirtualAd-dress	AElf.Types.Hash	The virtual address to use as sender.
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
methodName	System.String	The name of method you want to invoke.
message	Google.Protobuf.ByteString	The protobuf message that will be the input to the call.

SmartContractConstants `type`

Namespace

AElf.Sdk.CSharp

Summary

Static class containing the hashes built from the names of the contracts.

StringState `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around string values for use in smart contract state.

UInt32State `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around unsigned 32-bit integer values for use in smart contract state.

UInt64State `type`

Namespace

AElf.Sdk.CSharp.State

Summary

Wrapper around unsigned 64-bit integer values for use in smart contract state.

19.2 AElf.CSharp.Core

19.2.1 Contents

  - *Max(timestamp1,timestamp2)*
  - *Milliseconds(duration)*
- *UnaryServerMethod*

Builder `type`

Namespace

AElf.CSharp.Core.ServerServiceDefinition

Summary

Builder class for *ServerServiceDefinition*.

ctor() `constructor`

Summary

Creates a new instance of builder.

Parameters

This constructor has no parameters.

AddMethod“2(method,handler) `method`

Summary

Adds a definition for a single request - single response method.

Returns

This builder instance.

Parameters

Name	Type	Description
method	*AElf.CSharp.Core.Method*	The method.
handler	*AElf.CSharp.Core.UnaryServerMethod*	The method handler.

Generic Types

Name	Description
TRequest	The request message class.
TResponse	The response message class.

Build() `method`

Summary

Creates an immutable `ServerServiceDefinition` from this builder.

Returns

The `ServerServiceDefinition` object.

Parameters

This method has no parameters.

EncodingHelper `type`

Namespace

AElf.CSharp.Core.Utils

Summary

Helper class for serializing strings.

EncodeUtf8(str) `method`

Summary

Serializes a UTF-8 string to a byte array.

Returns

the serialized string.

Parameters

Name	Type	Description
str	System.String	

IMethod `type`

Namespace

AElf.CSharp.Core

Summary

A non-generic representation of a remote method.

FullName `property`

Summary

Gets the fully qualified name of the method. On the server side, methods are dispatched based on this name.

Name `property`

Summary

Gets the unqualified name of the method.

ServiceName `property`

Summary

Gets the name of the service to which this method belongs.

Type `property`

Summary

Gets the type of the method.

Marshaller `type`

Namespace

AElf.CSharp.Core

Summary

Encapsulates the logic for serializing and deserializing messages.

ctor(serializer,deserializer) `constructor`

Summary

Initializes a new marshaller from simple serialize/deserialize functions.

Parameters

Name	Type	Description
serializer	System.Func	Function that will be used to deserialize messages.

Deserializer `property`

Summary

Gets the deserializer function.

Serializer `property`

Summary

Gets the serializer function.

Marshallers `type`

Namespace

AElf.CSharp.Core

Summary

Utilities for creating marshallers.

StringMarshaller `property`

Summary

Returns a marshaller for `string` type. This is useful for testing.

Create() `method`

Summary

Creates a marshaller from specified serializer and deserializer.

Parameters

This method has no parameters.

MethodType `type`

Namespace

AElf.CSharp.Core

Action `constants`

Summary

The method modifies the contrac state.

View `constants`

Summary

The method doesn't modify the contract state.

Method `type`

Namespace

AElf.CSharp.Core

Summary

A description of a remote method.

Generic Types

Name	Description
TRequest	Request message type for this method.
TResponse	Response message type for this method.

ctor(type,serviceName,name,requestMarshaller,responseMarshaller) `constructor`

Summary

Initializes a new instance of the `Method` class.

Parameters

Name	Type	Description
type	*AElf.CSharp.Core.Method*	Type of method.
serviceName	System.String	Name of service this method belongs to.
name	System.String	Unqualified name of the method.
request Marshaller	*AElf.CSharp.Core.Marshaller*	Marshaller used for request messages.
response Marshaller	*AElf.CSharp.Core.Marshaller*	Marshaller used for response messages.

FullName `property`

Summary

Gets the fully qualified name of the method. On the server side, methods are dispatched based on this name.

Name `property`

Summary

Gets the unqualified name of the method.

RequestMarshaller `property`

Summary

Gets the marshaller used for request messages.

ResponseMarshaller `property`

Summary

Gets the marshaller used for response messages.

ServiceName `property`

Summary

Gets the name of the service to which this method belongs.

Type `property`

Summary

Gets the type of the method.

GetFullName() `method`

Summary

Gets full name of the method including the service name.

Parameters

This method has no parameters.

Preconditions `type`

Namespace

AElf.CSharp.Core.Utils

CheckNotNull(reference) `method`

Summary

Throws ArgumentNullException if reference is null.

Parameters

Name	Type	Description
reference		The reference.

CheckNotNull(reference,paramName) `method`

Summary

Throws ArgumentNullException if reference is null.

Parameters

Name	Type	Description
reference		The reference.
paramName	System.String	The parameter name.

SafeMath `type`

Namespace

AElf.CSharp.Core

Summary

Helper methods for safe math operations that explicitly check for overflow.

ServerServiceDefinition `type`

Namespace

AElf.CSharp.Core

Summary

Stores mapping of methods to server call handlers. Normally, the `ServerServiceDefinition` objects will be created by the `BindService` factory method that is part of the autogenerated code for a protocol buffers service definition.

BindService() `method`

Summary

Forwards all the previously stored `AddMethod` calls to the service binder.

Parameters

This method has no parameters.

CreateBuilder() `method`

Summary

Creates a new builder object for `ServerServiceDefinition`.

Returns

The builder object.

Parameters

This method has no parameters.

ServiceBinderBase `type`

Namespace

AElf.CSharp.Core

Summary

Allows binding server-side method implementations in alternative serving stacks. Instances of this class are usually populated by the `BindService` method that is part of the autogenerated code for a protocol buffers service definition.

AddMethod(method,handler) `method`

Summary

Adds a definition for a single request - single response method.

Parameters

Name	Type	Description
method	*AElf.CSharp.Core.Method*	The method.
handler	*AElf.CSharp.Core.UnaryServerMethod*	The method handler.

Generic Types

Name	Description
TRequest	The request message class.
TResponse	The response message class.

TimestampExtensions `type`

Namespace

AElf.CSharp.Core.Extension

Summary

Helper methods for dealing with protobuf timestamps.

AddDays(timestamp,days) `method`

Summary

Adds a given amount of days to a timestamp. Returns a new instance.

Returns

a new timestamp instance.

Parameters

Name	Type	Description
timestamp	Google.Protobuf.WellKnown Types.Timestamp	the timestamp.
days	System. Int64	the amount of days.

AddHours(timestamp,hours) `method`

Summary

Adds a given amount of hours to a timestamp. Returns a new instance.

Returns

a new timestamp instance.

Parameters

Name	Type	Description
timestamp	Google.Protobuf .WellKnownTypes.Timestamp	the timestamp.
hours	System.Int64	the amount of hours.

AddMilliseconds(timestamp,milliseconds) `method`

Summary

Adds a given amount of milliseconds to a timestamp. Returns a new instance.

Returns

a new timestamp instance.

Parameters

Name	Type	Description
timestamp	Google.Protobuf. WellKnownTypes.Timestamp	the timestamp.
milliseconds	System. Int64	the amount of milliseconds to add.

AddMinutes(timestamp,minutes) `method`

Summary

Adds a given amount of minutes to a timestamp. Returns a new instance.

Returns

a new timestamp instance.

Parameters

Name	Type	Description
timestamp	Google.Protobuf .WellKnownTypes.Timestamp	the timestamp.
minutes	System.Int64	the amount of minutes.

AddSeconds(timestamp,seconds) `method`

Summary

Adds a given amount of seconds to a timestamp. Returns a new instance.

Returns

a new timestamp instance.

Parameters

Name	Type	Description
timestamp	Google.Protobuf .WellKnownTypes.Timestam	the timestamp.
seconds	System.Int64	the amount of seconds.

Max(timestamp1,timestamp2) `method`

Summary

Compares two timestamps and returns the greater one.

Returns

the greater timestamp.

Parameters

Name	Type	Description
timestamp1	Google.Protobuf .WellKnownTypes.Timestamp	the first timestamp
timestamp2	Google.Protobuf .WellKnownTypes.Timestamp	the second timestamp

Milliseconds(duration) `method`

Summary

Converts a protobuf duration to long.

Returns

the duration represented with a long.

Parameters

Name	Type	Description
duration	Google.Protobuf. WellKnownTypes.Duration	the duration to convert.

UnaryServerMethod `type`

Namespace

AElf.CSharp.Core

Summary

Handler for a contract method.

Generic Types

Name	Description
TRequest	Request message type for this method.
TResponse	Response message type for this method.

CHAPTER 20

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**).

20.1 AElf.Contracts.Association

Association contract.

Organizations established to achieve specific goals can use this contract to cooperatively handle transactions within the organization

Implement AElf Standards ACS1 and ACS3.

20.1.1 Contract Methods

Method Name	Request Type	Response Type	Description
CreateOrganization	*Association.CreateOrganizationInput*	*aelf.Address*	Create an organization and return its address.
CreateOrganizationBySystemContract	*Association.CreateOrganizationBySystemContractInput*	*aelf.Address*	Creates an organization by system contract and return its address.
AddMember	*aelf.Address*	*google.protobuf.Empty*	Add organization members.
RemoveMember	*aelf.Address*	*google.protobuf.Empty*	Remove organization members.
ChangeMember	*Association.ChangeMemberInput*	*google.protobuf.Empty*	Replace organization member with a new member.
GetOrganization	*aelf.Address*	*Association.Organization*	Get the organization according to the organization address.
CalculateOrganizationAddress	*Association.CreateOrganizationInput*	*aelf.Address*	Calculate the input and return the organization address.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS3

Method Name	Request Type	Response Type	Description
CreateProposal	*acs3.CreateProposalInput*	*aelf.Hash*	Create a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract. Return id of the newly created proposal.
Approve	*aelf.Hash*	*google.protobuf.Empty*	Approve a proposal according to the proposal ID.
Reject	*aelf.Hash*	*google.protobuf.Empty*	Reject a proposal according to the proposal ID.
Abstain	*aelf.Hash*	*google.protobuf.Empty*	Abstain a proposal according to the proposal ID.
Release	*aelf.Hash*	*google.protobuf.Empty*	Release a proposal according to the proposal ID and send a transaction to the specified contract.
ChangeOrganizationThreshold	*acs3.ProposalReleaseThreshold*	*google.protobuf.Empty*	Change 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.
ChangeOrganizationProposerWhiteList	*acs3.ProposerWhiteList*	*google.protobuf.Empty*	Change the white list of organization proposer. This method overrides the list of whitelisted proposers.
CreateProposalBySystemContract	*acs3.CreateProposalBySystemContractInput*	*aelf.Hash*	Create a proposal by system contracts, and return id of the newly created proposal.
ClearProposal	*aelf.Hash*	*google.protobuf.Empty*	Remove the specified proposal. If the proposal is in effect, the cleanup fails.
GetProposal	*aelf.Hash*	*acs3.ProposalOutput*	Get the proposal according to the proposal ID.
ValidateOrganizationExist	*aelf.Address*	*google.protobuf.BoolValue*	Check the existence of an organization.
ValidateProposerInWhiteList	*acs3.ValidateProposerInWhiteListInput*	*google.protobuf.BoolValue*	Check if the proposer is whitelisted.

20.1.2 Contract Types

AElf.Contracts.Association

Association.ChangeMemberInput

Field	Type	Description	Label
old_member	*aelf.Address*	The old member address.	
new_member	*aelf.Address*	The new member address.	

Association.CreateOrganizationBySystemContractInput

Field	Type	Description	Label
organization_creation_input	*CreateOrganizationInput*	The parameters of creating organization.	
organization_address_feedback_method	*string*	The organization address callback method which replies the organization address to caller contract.	

Association.CreateOrganizationInput

Field	Type	Description	Label
organization_member_list	*OrganizationMemberList*	Initial organization members.	
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
proposer_white_list	*acs3.ProposerWhiteList*	The proposer whitelist.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Association.MemberAdded

Field	Type	Description	Label
member	*aelf.Address*	The added member address.	
organization_address	*aelf.Address*	The organization address.	

Association.MemberChanged

Field	Type	Description	Label
old_member	*aelf.Address*	The old member address.	
new_member	*aelf.Address*	The new member address.	
organization_address	*aelf.Address*	The organization address.	

Association.MemberRemoved

Field	Type	Description	Label
member	*aelf.Address*	The removed member address.	
organization_address	*aelf.Address*	The organization address.	

Association.Organization

Field	Type	Description	Label
organization_member_list	*OrganizationMemberList*	The organization members.	
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
proposer_white_list	*acs3.ProposerWhiteList*	The proposer whitelist.	
organization_address	*aelf.Address*	The address of organization.	
organization_hash	*aelf.Hash*	The organizations id.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Association.OrganizationMemberList

Field	Type	Description	Label
organization_members	*aelf.Address*	The address of organization members.	repeated

Association.ProposalInfo

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal ID.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
organization_address	*aelf.Address*	The address of this proposals organization.	
approvals	*aelf.Address*	Address list of approved.	repeated
rejections	*aelf.Address*	Address list of rejected.	repeated
abstentions	*aelf.Address*	Address list of abstained.	repeated
proposal_description_url	*string*	Url is used for proposal describing.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS3

acs3.CreateProposalBySystemContractInput

Field	Type	Description	Label
proposal_input	*CreateProposalInput*	The parameters of creating proposal.	
origin_proposer	*aelf.Address*	The actor that trigger the call.	

acs3.CreateProposalInput

Field	Type	Description	Label
contract_method_name	*string*	The name of the method to call after release.	
to_address	*aelf.Address*	The address of the contract to call after release.	
params	*bytes*	The parameter of the method to be called after the release.	
expired_time	*google.protobuf.Timestamp*	The timestamp at which this proposal will expire.	
organization_address	*aelf.Address*	The address of the organization.	
proposal_description_url	*string*	Url is used for proposal describing.	
token	*aelf.Hash*	The token is for proposal id generation and with this token, proposal id can be calculated before proposing.	

acs3.OrganizationCreated

Field	Type	Description	Label
organization_address	*aelf.Address*	The address of the created organization.	

acs3.OrganizationHashAddressPair

Field	Type	Description	Label
organization_hash	*aelf.Hash*	The id of organization.	
organization_address	*aelf.Address*	The address of organization.	

acs3.OrganizationThresholdChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address	
proposer_release_threshold	*ProposalReleaseThreshold*	The new release threshold.	

acs3.OrganizationWhiteListChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address.	
proposer_white_list	*ProposerWhiteList*	The new proposer whitelist.	

acs3.ProposalCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the created proposal.	
organization_address	*aelf.Address*	The organization address of the created proposal.	

acs3.ProposalOutput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
organization_address	*aelf.Address*	The address of this proposals organization.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
to_be_released	*bool*	Indicates if this proposal is releasable.	
approval_count	*int64*	Approval count for this proposal.	
rejection_count	*int64*	Rejection count for this proposal.	
abstention_count	*int64*	Abstention count for this proposal.	

acs3.ProposalReleaseThreshold

Field	Type	Description	Label
minimal_approval_threshold	*int64*	The value for the minimum approval threshold.	
maximal_rejection_threshold	*int64*	The value for the maximal rejection threshold.	
maximal_abstention_threshold	*int64*	The value for the maximal abstention threshold.	
minimal_vote_threshold	*int64*	The value for the minimal vote threshold.	

acs3.ProposalReleased

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the released proposal.	
organization_address	*aelf.Address*	The organization address of the released proposal.	

acs3.ProposerWhiteList

Field	Type	Description	Label
proposers	*aelf.Address*	The address of the proposers	repeated

acs3.ReceiptCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
address	*aelf.Address*	The sender address.	
receipt_type	*string*	The type of receipt(Approve, Reject or Abstain).	
time	*google.protobuf.Timestamp*	The timestamp of this method call.	
organization_address	*aelf.Address*	The address of the organization.	

acs3.ValidateProposerInWhiteListInput

Field	Type	Description	Label
proposer	*aelf.Address*	The address to search/check.	
organization_address	*aelf.Address*	The address of the organization.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.2 AElf.Contracts.Referendum

Referendum contract.

Production nodes or associations cannot determine all decisions. Some extremely important decisions, especially those involving user rights and interests, should involve all users and give full control to the user's voting for governance. The Referendum contract is built for this.

Implement AElf Standards ACS1 and ACS3.

20.2.1 Contract Methods

Method Name	Request Type	Response Type	Description
ReclaimVoteToken	*aelf.Hash*	*google.protobuf.Empty*	Unlock the token used for voting according to proposal id.
CreateOrganization	*Referendum.CreateOrganizationInput*	*aelf.Address*	Create an organization and return its address.
CreateOrganizationBySystemContract	*Referendum.CreateOrganizationBySystemContractInput*	*aelf.Address*	Creates an organization by system contract and return its address.
GetOrganization	*aelf.Address*	*Referendum.Organization*	Get the organization according to the organization address.
CalculateOrganizationAddress	*Referendum.CreateOrganizationInput*	*aelf.Address*	Calculate the input and return the organization address.
GetProposalVirtualAddress	*aelf.Hash*	*aelf.Address*	Get the virtual address of a proposal based on the proposal id.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS3

Method Name	Request Type	Response Type	Description
CreateProposal	*acs3.CreateProposalInput*	*aelf.Hash*	Create a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract. Return id of the newly created proposal.
Approve	*aelf.Hash*	*google.protobuf.Empty*	Approve a proposal according to the proposal ID.
Reject	*aelf.Hash*	*google.protobuf.Empty*	Reject a proposal according to the proposal ID.
Abstain	*aelf.Hash*	*google.protobuf.Empty*	Abstain a proposal according to the proposal ID.
Release	*aelf.Hash*	*google.protobuf.Empty*	Release a proposal according to the proposal ID and send a transaction to the specified contract.
ChangeOrganizationThreshold	*acs3.ProposalReleaseThreshold*	*google.protobuf.Empty*	Change 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.
ChangeOrganizationProposerWhiteList	*acs3.ProposerWhiteList*	*google.protobuf.Empty*	Change the white list of organization proposer. This method overrides the list of whitelisted proposers.
CreateProposalBySystemContract	*acs3.CreateProposalBySystemContractInput*	*aelf.Hash*	Create a proposal by system contracts, and return id of the newly created proposal.
ClearProposal	*aelf.Hash*	*google.protobuf.Empty*	Remove the specified proposal. If the proposal is in effect, the cleanup fails.
GetProposal	*aelf.Hash*	*acs3.ProposalOutput*	Get the proposal according to the proposal ID.
ValidateOrganizationExist	*aelf.Address*	*google.protobuf.BoolValue*	Check the existence of an organization.
ValidateProposerInWhiteList	*acs3.ValidateProposerInWhiteListInput*	*google.protobuf.BoolValue*	Check if the proposer is whitelisted.

20.2.2 Contract Types

AElf.Contracts.Referendum

Referendum.CreateOrganizationBySystemContractInput

Field	Type	Description	Label
organization_creation_input	*CreateOrganizationInput*	The parameters of creating organization.	
organization_address_feedback_method	*string*	The organization address callback method which replies the organization address to caller contract.	

Referendum.CreateOrganizationInput

Field	Type	Description	Label
token_symbol	*string*	The token used during proposal operations.	
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
proposer_white_list	*acs3.ProposerWhiteList*	The proposer whitelist.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Referendum.Organization

Field	Type	Description	Label
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
token_symbol	*string*	The token used during proposal operations.	
organization_address	*aelf.Address*	The address of organization.	
organization_hash	*aelf.Hash*	The organizations id.	
proposer_white_list	*acs3.ProposerWhiteList*	The proposer whitelist.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Referendum.ProposalInfo

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal ID.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
organization_address	*aelf.Address*	The address of this proposals organization.	
approval_count	*int64*	The count of approved.	
rejection_count	*int64*	The count of rejected.	
abstention_count	*int64*	The count of abstained.	
proposal_description_url	*string*	Url is used for proposal describing.	

Referendum.Receipt

Field	Type	Description	Label
amount	*int64*	The amount of token locked.	
token_symbol	*string*	The symbol of token locked.	
lock_id	*aelf.Hash*	The lock id.	

Referendum.ReferendumReceiptCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
address	*aelf.Address*	The sender address.	
symbol	*string*	The symbol of token locked.	
amount	*int64*	The amount of token locked.	
receipt_type	*string*	The type of receipt(Approve, Reject or Abstain).	
time	*google.protobuf.Timestamp*	The timestamp of this method call.	
organization_address	*aelf.Address*	The address of the organization.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS3

acs3.CreateProposalBySystemContractInput

Field	Type	Description	Label
proposal_input	*CreateProposalInput*	The parameters of creating proposal.	
origin_proposer	*aelf.Address*	The actor that trigger the call.	

acs3.CreateProposalInput

Field	Type	Description	Label
contract_method_name	*string*	The name of the method to call after release.	
to_address	*aelf.Address*	The address of the contract to call after release.	
params	*bytes*	The parameter of the method to be called after the release.	
expired_time	*google.protobuf.Timestamp*	The timestamp at which this proposal will expire.	
organization_address	*aelf.Address*	The address of the organization.	
proposal_description_url	*string*	Url is used for proposal describing.	
token	*aelf.Hash*	The token is for proposal id generation and with this token, proposal id can be calculated before proposing.	

acs3.OrganizationCreated

Field	Type	Description	Label
organization_address	*aelf.Address*	The address of the created organization.	

acs3.OrganizationHashAddressPair

Field	Type	Description	Label
organization_hash	*aelf.Hash*	The id of organization.	
organization_address	*aelf.Address*	The address of organization.	

acs3.OrganizationThresholdChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address	
proposer_release_threshold	*ProposalReleaseThreshold*	The new release threshold.	

acs3.OrganizationWhiteListChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address.	
proposer_white_list	*ProposerWhiteList*	The new proposer whitelist.	

acs3.ProposalCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the created proposal.	
organization_address	*aelf.Address*	The organization address of the created proposal.	

acs3.ProposalOutput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
organization_address	*aelf.Address*	The address of this proposals organization.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
to_be_released	*bool*	Indicates if this proposal is releasable.	
approval_count	*int64*	Approval count for this proposal.	
rejection_count	*int64*	Rejection count for this proposal.	
abstention_count	*int64*	Abstention count for this proposal.	

acs3.ProposalReleaseThreshold

Field	Type	Description	Label
minimal_approval_threshold	*int64*	The value for the minimum approval threshold.	
maximal_rejection_threshold	*int64*	The value for the maximal rejection threshold.	
maximal_abstention_threshold	*int64*	The value for the maximal abstention threshold.	
minimal_vote_threshold	*int64*	The value for the minimal vote threshold.	

acs3.ProposalReleased

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the released proposal.	
organization_address	*aelf.Address*	The organization address of the released proposal.	

acs3.ProposerWhiteList

Field	Type	Description	Label
proposers	*aelf.Address*	The address of the proposers	repeated

acs3.ReceiptCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
address	*aelf.Address*	The sender address.	
receipt_type	*string*	The type of receipt(Approve, Reject or Abstain).	
time	*google.protobuf.Timestamp*	The timestamp of this method call.	
organization_address	*aelf.Address*	The address of the organization.	

acs3.ValidateProposerInWhiteListInput

Field	Type	Description	Label
proposer	*aelf.Address*	The address to search/check.	
organization_address	*aelf.Address*	The address of the organization.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.3 AElf.Contracts.Parliament

Parliament contract.

The production nodes use the Parliament contract to govern important matters. In the initial state, the production nodes are members of the parliament, and only when two-thirds of the production nodes vote in favor of a given decision, will it be executed.

Implement AElf Standards ACS1 and ACS3.

20.3.1 Contract Methods

Method Name	Request Type	Response Type	Description
Initialize	*Parliament.InitializeInput*	*google.protobuf.Empty*	Initialize parliament proposer whitelist and create the first parliament organization with specific proposer_authority_required.
CreateOrganization	*Parliament.CreateOrganizationInput*	*aelf.Address*	Create an organization and return its address.
ApproveMultiProposals	*Parliament.ProposalIdList*	*google.protobuf.Empty*	Batch approval proposal.
CreateOrganizationBySystemContract	*Parliament.CreateOrganizationBySystemContractInput*	*aelf.Address*	Creates an organization by system contract and return its address.
GetOrganization	*aelf.Address*	*Parliament.Organization*	Get the organization according to the organization address.
GetDefaultOrganizationAddress	*google.protobuf.Empty*	*aelf.Address*	Get the default organization address.
ValidateAddressIsParliamentMember	*aelf.Address*	*google.protobuf.BoolValue*	Validates if the provided address is a parliament member.
GetProposerWhiteList	*google.protobuf.Empty*	*acs3.ProposerWhiteList*	Returns the list of whitelisted proposers.
GetNotVotedPendingProposals	*Parliament.ProposalIdList*	*Parliament.ProposalIdList*	Filter still pending ones not yet voted by the sender from provided proposals.
GetNotVotedProposals	*Parliament.ProposalIdList*	*Parliament.ProposalIdList*	Filter not yet voted ones by the sender from provided proposals.
CalculateOrganizationAddress	*Parliament.CreateOrganizationInput*	*aelf.Address*	Calculates with input and return the organization address.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS3

Method Name	Request Type	Response Type	Description
CreateProposal	*acs3.CreateProposalInput*	*aelf.Hash*	Create a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract. Return id of the newly created proposal.
Approve	*aelf.Hash*	*google.protobuf.Empty*	Approve a proposal according to the proposal ID.
Reject	*aelf.Hash*	*google.protobuf.Empty*	Reject a proposal according to the proposal ID.
Abstain	*aelf.Hash*	*google.protobuf.Empty*	Abstain a proposal according to the proposal ID.
Release	*aelf.Hash*	*google.protobuf.Empty*	Release a proposal according to the proposal ID and send a transaction to the specified contract.
ChangeOrganizationThreshold	*acs3.ProposalReleaseThreshold*	*google.protobuf.Empty*	Change 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.
ChangeOrganizationProposerWhiteList	*acs3.ProposerWhiteList*	*google.protobuf.Empty*	Change the white list of organization proposer. This method overrides the list of whitelisted proposers.
CreateProposalBySystemContract	*acs3.CreateProposalBySystemContractInput*	*aelf.Hash*	Create a proposal by system contracts, and return id of the newly created proposal.
ClearProposal	*aelf.Hash*	*google.protobuf.Empty*	Remove the specified proposal. If the proposal is in effect, the cleanup fails.
GetProposal	*aelf.Hash*	*acs3.ProposalOutput*	Get the proposal according to the proposal ID.
ValidateOrganizationExist	*aelf.Address*	*google.protobuf.BoolValue*	Check the existence of an organization.
ValidateProposerInWhiteList	*acs3.ValidateProposerInWhiteListInput*	*google.protobuf.BoolValue*	Check if the proposer is whitelisted.

20.3.2 Contract Types

AElf.Contracts.Parliament

Parliament.CreateOrganizationBySystemContractInput

Field	Type	Description	Label
organization_creation_input	*CreateOrganizationInput*	The parameters of creating organization.	
organization_address_feedback_method	*string*	The organization address callback method which replies the organization address to caller contract.	

Parliament.CreateOrganizationInput

Field	Type	Description	Label
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
proposer_authority_required	*bool*	Setting this to true can allow anyone to create proposals.	
parliament_member_proposing_allowed	*bool*	Setting this to true can allow parliament member to create proposals.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Parliament.InitializeInput

Field	Type	Description	Label
privileged_proposer	*aelf.Address*	Privileged proposer would be the first address in parliament proposer whitelist.	
proposer_authority_required	*bool*	The setting indicates if proposals need authority to be created for first/default parliament organization.	

Parliament.Organization

Field	Type	Description	Label
proposer_authority_required	*bool*	Indicates if proposals need authority to be created.	
organization_address	*aelf.Address*	The organization address.	
organization_hash	*aelf.Hash*	The organization id.	
proposal_release_threshold	*acs3.ProposalReleaseThreshold*	The threshold for releasing the proposal.	
parliament_member_proposing_allowed	*bool*	Indicates if parliament member can propose to this organization.	
creation_token	*aelf.Hash*	The creation token is for organization address generation.	

Parliament.ProposalIdList

Field	Type	Description	Label
proposal_ids	*aelf.Hash*	The list of proposal ids.	repeated

Parliament.ProposalInfo

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal ID.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
organization_address	*aelf.Address*	The address of this proposals organization.	
approvals	*aelf.Address*	Address list of approved.	repeated
rejections	*aelf.Address*	Address list of rejected.	repeated
abstentions	*aelf.Address*	Address list of abstained.	repeated
proposal_description_url	*string*	Url is used for proposal describing.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS3

acs3.CreateProposalBySystemContractInput

Field	Type	Description	Label
proposal_input	*CreateProposalInput*	The parameters of creating proposal.	
origin_proposer	*aelf.Address*	The actor that trigger the call.	

acs3.CreateProposalInput

Field	Type	Description	Label
contract_method_name	*string*	The name of the method to call after release.	
to_address	*aelf.Address*	The address of the contract to call after release.	
params	*bytes*	The parameter of the method to be called after the release.	
expired_time	*google.protobuf.Timestamp*	The timestamp at which this proposal will expire.	
organization_address	*aelf.Address*	The address of the organization.	
proposal_description_url	*string*	Url is used for proposal describing.	
token	*aelf.Hash*	The token is for proposal id generation and with this token, proposal id can be calculated before proposing.	

acs3.OrganizationCreated

Field	Type	Description	Label
organization_address	*aelf.Address*	The address of the created organization.	

acs3.OrganizationHashAddressPair

Field	Type	Description	Label
organization_hash	*aelf.Hash*	The id of organization.	
organization_address	*aelf.Address*	The address of organization.	

acs3.OrganizationThresholdChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address	
proposer_release_threshold	*ProposalReleaseThreshold*	The new release threshold.	

acs3.OrganizationWhiteListChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address.	
proposer_white_list	*ProposerWhiteList*	The new proposer whitelist.	

acs3.ProposalCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the created proposal.	
organization_address	*aelf.Address*	The organization address of the created proposal.	

acs3.ProposalOutput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
organization_address	*aelf.Address*	The address of this proposals organization.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
to_be_released	*bool*	Indicates if this proposal is releasable.	
approval_count	*int64*	Approval count for this proposal.	
rejection_count	*int64*	Rejection count for this proposal.	
abstention_count	*int64*	Abstention count for this proposal.	

acs3.ProposalReleaseThreshold

Field	Type	Description	Label
minimal_approval_threshold	*int64*	The value for the minimum approval threshold.	
maximal_rejection_threshold	*int64*	The value for the maximal rejection threshold.	
maximal_abstention_threshold	*int64*	The value for the maximal abstention threshold.	
minimal_vote_threshold	*int64*	The value for the minimal vote threshold.	

acs3.ProposalReleased

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the released proposal.	
organization_address	*aelf.Address*	The organization address of the released proposal.	

acs3.ProposerWhiteList

Field	Type	Description	Label
proposers	*aelf.Address*	The address of the proposers	repeated

acs3.ReceiptCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
address	*aelf.Address*	The sender address.	
receipt_type	*string*	The type of receipt(Approve, Reject or Abstain).	
time	*google.protobuf.Timestamp*	The timestamp of this method call.	
organization_address	*aelf.Address*	The address of the organization.	

acs3.ValidateProposerInWhiteListInput

Field	Type	Description	Label
proposer	*aelf.Address*	The address to search/check.	
organization_address	*aelf.Address*	The address of the organization.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.4 AElf.Contracts.Consensus.AEDPoS

AEDPoS contract.

Used to managing block producers and synchronizing data.

Implement AElf Standards ACS1, ACS4, ACS6, ACS10 and ACS11.

20.4.1 Contract Methods

Method Name	Request Type	Response Type
InitialAElfConsensusContract	*AEDPoS.InitialAElfConsensusContractInput*	*google.protobuf.Empty*
FirstRound	*AEDPoS.Round*	*google.protobuf.Empty*
UpdateValue	*AEDPoS.UpdateValueInput*	*google.protobuf.Empty*
NextRound	*AEDPoS.Round*	*google.protobuf.Empty*
NextTerm	*AEDPoS.Round*	*google.protobuf.Empty*
UpdateTinyBlockInformation	*AEDPoS.TinyBlockInput*	*google.protobuf.Empty*
SetMaximumMinersCount	*google.protobuf.Int32Value*	*google.protobuf.Empty*
ChangeMaximumMinersCountController	*AuthorityInfo*	*google.protobuf.Empty*
RecordCandidateReplacement	*AEDPoS.RecordCandidateReplacementInput*	*google.protobuf.Empty*
GetCurrentMinerList	*google.protobuf.Empty*	*AEDPoS.MinerList*
GetCurrentMinerPubkeyList	*google.protobuf.Empty*	*AEDPoS.PubkeyList*
GetCurrentMinerListWithRoundNumber	*google.protobuf.Empty*	*AEDPoS.MinerListWithRoundNumber*
GetRoundInformation	*google.protobuf.Int64Value*	*AEDPoS.Round*
GetCurrentRoundNumber	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetCurrentRoundInformation	*google.protobuf.Empty*	*AEDPoS.Round*
GetPreviousRoundInformation	*google.protobuf.Empty*	*AEDPoS.Round*
GetCurrentTermNumber	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetCurrentTermMiningReward	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetMinerList	*AEDPoS.GetMinerListInput*	*AEDPoS.MinerList*
GetPreviousMinerList	*google.protobuf.Empty*	*AEDPoS.MinerList*
GetMinedBlocksOfPreviousTerm	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetNextMinerPubkey	*google.protobuf.Empty*	*google.protobuf.StringValue*
IsCurrentMiner	*aelf.Address*	*google.protobuf.BoolValue*
GetNextElectCountDown	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetPreviousTermInformation	*google.protobuf.Int64Value*	*AEDPoS.Round*
GetRandomHash	*google.protobuf.Int64Value*	*aelf.Hash*
GetMaximumBlocksCount	*google.protobuf.Empty*	*google.protobuf.Int32Value*
GetMaximumMinersCount	*google.protobuf.Empty*	*google.protobuf.Int32Value*
GetMaximumMinersCountController	*google.protobuf.Empty*	*AuthorityInfo*
GetMainChainCurrentMinerList	*google.protobuf.Empty*	*AEDPoS.MinerList*
GetPreviousTermMinerPubkeyList	*google.protobuf.Empty*	*AEDPoS.PubkeyList*

Table 1 – continu

Method Name	Request Type	Response Type
GetCurrentMiningRewardPerBlock	*google.protobuf.Empty*	*google.protobuf.Int64Value*
SetMinerIncreaseInterval	*google.protobuf.Int64Value*	*google.protobuf.Empty*
GetMinerIncreaseInterval	*google.protobuf.Empty*	*google.protobuf.Int64Value*

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS4

Method Name	Request Type	Response Type	Description
GetConsensusCommand	*google.protobuf.BytesValue*	*acs4.ConsensusCommand*	Generate a consensus command based on the consensus contract state and the input public key.
GetConsensusExtraData	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Generate consensus extra data when a block is generated.
GenerateConsensusTransactions	*google.protobuf.BytesValue*	*acs4.TransactionList*	Generate consensus system transactions 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	*google.protobuf.BytesValue*	*acs4.ValidationResult*	Before executing the block, verify that the consensus information in the block header is correct.
ValidateConsensusAfterExecution	*google.protobuf.BytesValue*	*acs4.ValidationResult*	After executing the block, verify that the state information written to the consensus is correct.

AElf.Standards.ACS6

Method Name	Request Type	Response Type	Description
GetRandomBytes	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Get random number according to block height.

AElf.Standards.ACS10

Method Name	Request Type	Response Type	Description
Donate	*acs10.DonateInput*	*google.protobuf.Empty*	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.
Release	*acs10.ReleaseInput*	*google.protobuf.Empty*	Release dividend pool according the period number.
SetSymbolList	*acs10.SymbolList*	*google.protobuf.Empty*	Set the token symbols dividend pool supports.
GetSymbolList	*google.protobuf.Empty*	*acs10.SymbolList*	Query the token symbols dividend pool supports.
GetUndistributedDividends	*google.protobuf.Empty*	*acs10.Dividends*	Query the balance of undistributed tokens whose symbols are included in the symbol list.
GetDividends	*google.protobuf.Int64Value*	*acs10.Dividends*	Query the dividend information according to the height.

AElf.Standards.ACS11

Method Name	Request Type	Response Type	Description
UpdateInformationFromCrossChain	*google.protobuf.BytesValue*	*google.protobuf.Empty*	Update the consensus information of the side chain.
GetChainInitializationInformation	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Get the current miner list and consensus round information.
CheckCrossChainIndexingPermission	*aelf.Address*	*google.protobuf.BoolValue*	Verify that the input address is the current miner.

20.4.2 Contract Types

AElf.Contracts.Consensus.AEDPoS

AEDPoS.AElfConsensusHeaderInformation

Field	Type	Description	Label
sender_pubkey	*bytes*	The sender public key.	
round	*Round*	The round information.	
behaviour	*AElfConsensusBehaviour*	The behaviour of consensus.	

AEDPoS.AElfConsensusHint

Field	Type	Description	Label
behaviour	*AElfConsensusBehaviour*	The behaviour of consensus.	
round_id	*int64*	The round id.	
previous_round_id	*int64*	The previous round id.	

AEDPoS.AElfConsensusTriggerInformation

Field	Type	Description	Label
pubkey	*bytes*	The miner public key.	
in_value	*aelf.Hash*	The InValue for current round.	
previous_in_value	*aelf.Hash*	The InValue for previous round.	
behaviour	*AElfConsensusBehaviour*	The behaviour of consensus.	
encrypted_pieces	*AElfConsensusTriggerInformation.EncryptedPiecesEntry*	The encrypted pieces of InValue.	repeated
decrypted_pieces	*AElfConsensusTriggerInformation.DecryptedPiecesEntry*	The decrypted pieces of InValue.	repeated
revealed_in_values	*AElfConsensusTriggerInformation.RevealedInValuesEntry*	The revealed InValues.	repeated

AEDPoS.AElfConsensusTriggerInformation.DecryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.AElfConsensusTriggerInformation.EncryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.AElfConsensusTriggerInformation.RevealedInValuesEntry

Field	Type	Description	Label
key	*string*		
value	*aelf.Hash*		

AEDPoS.Candidates

Field	Type	Description	Label
pubkeys	*bytes*	The candidate public keys.	repeated

AEDPoS.ConsensusInformation

Field	Type	Description	Label
value	*bytes*		

AEDPoS.GetMinerListInput

Field	Type	Description	Label
term_number	*int64*	The term number.	

AEDPoS.HashList

Field	Type	Description	Label
values	*aelf.Hash*		repeated

AEDPoS.InitialAElfConsensusContractInput

Field	Type	Description	Label
is_term_stay_one	*bool*	Whether not to change the term.	
is_side_chain	*bool*	Is a side chain.	
period_seconds	*int64*	The number of seconds per term.	
miner_increase_interval	*int64*	The interval second that increases the number of miners.	

AEDPoS.IrreversibleBlockFound

Field	Type	Description	Label
irreversible_block_height	*int64*	The irreversible block height found.	

AEDPoS.IrreversibleBlockHeightUnacceptable

Field	Type	Description	Label
distance_to_irreversible_block_height	*int64*	Distance to the height of the last irreversible block.	

AEDPoS.LatestPubkeyToTinyBlocksCount

Field	Type	Description	Label
pubkey	*string*	The miner public key.	
blocks_count	*int64*	The count of blocks the miner produced.	

AEDPoS.MinerInRound

Field	Type	Description	Label
order	*int32*	The order of the miner producing block.	
is_extra_block_producer	*bool*	Is extra block producer in the current round.	
in_value	*aelf.Hash*	Generated by secret sharing and used for validation between miner.	
out_value	*aelf.Hash*	Calculated from current in value.	
signature	*aelf.Hash*	Calculated from current in value and signatures of previous round.	
expected_mining_time	*google.protobuf.Timestamp*	The expected mining time.	
produced_blocks	*int64*	The amount of produced blocks.	
missed_time_slots	*int64*	The amount of missed time slots.	
pubkey	*string*	The public key of this miner.	
previous_in_value	*aelf.Hash*	The InValue of the previous round.	
supposed_order_of_next_round	*int32*	The supposed order of mining for the next round.	
final_order_of_next_round	*int32*	The final order of mining for the next round.	
actual_mining_times	*google.protobuf.Timestamp*	The actual mining time, miners must fill actual mining time when they do the mining.	repeated
encrypted_pieces	*MinerInRound.EncryptedPiecesEntry*	The encrypted pieces of InValue.	repeated
decrypted_pieces	*MinerInRound.DecryptedPiecesEntry*	The decrypted pieces of InValue.	repeated
produced_tiny_blocks	*int64*	The amount of produced tiny blocks.	
implied_irreversible_block_height	*int64*	The irreversible block height that current miner recorded.	

AEDPoS.MinerInRound.DecryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.MinerInRound.EncryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.MinerList

Field	Type	Description	Label
pubkeys	*bytes*	The miners public key list.	repeated

AEDPoS.MinerListWithRoundNumber

Field	Type	Description	Label
miner_list	*MinerList*	The list of miners.	
round_number	*int64*	The round number.	

AEDPoS.MinerReplaced

Field	Type	Description	Label
new_miner_pubkey	*string*	The new miner public key.	

AEDPoS.MiningInformationUpdated

Field	Type	Description	Label
pubkey	*string*	The miner public key.	
mining_time	*google.protobuf.Timestamp*	The current block time.	
behaviour	*string*	The behaviour of consensus.	
block_height	*int64*	The current block height.	
previous_block_hash	*aelf.Hash*	The previous block hash.	

AEDPoS.MiningRewardGenerated

Field	Type	Description	Label
term_number	*int64*	The number of term the mining reward is generated.	
amount	*int64*	The amount of mining reward.	

AEDPoS.PubkeyList

Field	Type	Description	Label
pubkeys	*string*	The miners public key list.	repeated

AEDPoS.RandomNumberRequestInformation

Field	Type	Description	Label
target_round_number	*int64*	The random hash is likely generated during this round.	
order	*int64*		
expected_block_height	*int64*		

AEDPoS.RecordCandidateReplacementInput

Field	Type	Description	Label
old_pubkey	*string*		
new_pubkey	*string*		

AEDPoS.Round

Field	Type	Description	Label
round_number	*int64*	The round number.	
real_time_miners_information	*Round.RealTimeMinersInformationEntry*	Current miner information, miner public key -> miner information.	repeated
main_chain_miners_round_number	*int64*	The round number on the main chain	
blockchain_age	*int64*	The time from chain start to current round (seconds).	
extra_block_producer_of_previous_round	*string*	The miner public key that produced the extra block in the previous round.	
term_number	*int64*	The current term number.	
confirmed_irreversible_block_height	*int64*	The height of the confirmed irreversible block.	
confirmed_irreversible_block_round_number	*int64*	The round number of the confirmed irreversible block.	
is_miner_list_just_changed	*bool*	Is miner list different from the the miner list in the previous round.	
round_id_for_validation	*int64*	The round id, calculated by summing block producers' expecting time (second).	

AEDPoS.Round.RealTimeMinersInformationEntry

Field	Type	Description	Label
key	*string*		
value	*MinerInRound*		

AEDPoS.SecretSharingInformation

Field	Type	Description	Label
previous_round	*Round*	The previous round information.	
current_round_id	*int64*	The current round id.	
previous_round_id	*int64*	The previous round id.	

AEDPoS.TermInfo

Field	Type	Description	Label
term_number	*int64*		
round_number	*int64*		

AEDPoS.TermNumberLookUp

Field	Type	Description	Label
map	*TermNumberLookUp.MapEntry*	Term number -> Round number.	repeated

AEDPoS.TermNumberLookUp.MapEntry

Field	Type	Description	Label
key	*int64*		
value	*int64*		

AEDPoS.TinyBlockInput

Field	Type	Description	Label
round_id	*int64*	The round id.	
actual_mining_time	*google.protobuf.Timestamp*	The actual mining time.	
produced_blocks	*int64*	Count of blocks currently produced	

AEDPoS.UpdateValueInput

Field	Type	Description	Label
out_value	*aelf.Hash*	Calculated from current in value.	
signature	*aelf.Hash*	Calculated from current in value and signatures of previous round.	
round_id	*int64*	To ensure the values to update will be apply to correct round by comparing round id.	
previous_in_value	*aelf.Hash*	Publish previous in value for validation previous signature and previous out value.	
actual_mining_time	*google.protobuf.Timestamp*	The actual mining time, miners must fill actual mining time when they do the mining.	
supposed_order_of_next_round	*int32*	The supposed order of mining for the next round.	
tune_order_information	*UpdateValueInput.TuneOrderInformationEntry*	The tuning order of mining for the next round, miner public key -> order.	repeated
encrypted_pieces	*UpdateValueInput.EncryptedPiecesEntry*	The encrypted pieces of InValue.	repeated
decrypted_pieces	*UpdateValueInput.DecryptedPiecesEntry*	The decrypted pieces of InValue.	repeated
produced_blocks	*int64*	The amount of produced blocks.	
miners_previous_in_values	*UpdateValueInput.MinersPreviousInValuesEntry*	The InValue in the previous round, miner public key -> InValue.	repeated
implied_irreversible_block_height	*int64*	The irreversible block height that miner recorded.	

AEDPoS.UpdateValueInput.DecryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.UpdateValueInput.EncryptedPiecesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

AEDPoS.UpdateValueInput.MinersPreviousInValuesEntry

Field	Type	Description	Label
key	*string*		
value	*aelf.Hash*		

AEDPoS.UpdateValueInput.TuneOrderInformationEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

AEDPoS.VoteMinersCountInput

Field	Type	Description	Label
miners_count	*int32*		
amount	*int64*		

AEDPoS.AElfConsensusBehaviour

Name	Number	Description
UPDATE_VALUE	0	
NEXT_ROUND	1	
NEXT_TERM	2	
NOTHING	3	
TINY_BLOCK	4	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS4

acs4.ConsensusCommand

Field	Type	Description	Label
limit_milliseconds_of_mining_block	*int32*	Time limit of mining next block.	
hint	*bytes*	Context of Hint is diverse according to the consensus protocol we choose, so we use bytes.	
arranged_mining_time	*google.protobuf.Timestamp*	The time of arrange mining.	
mining_due_time	*google.protobuf.Timestamp*	The expiration time of mining.	

acs4.TransactionList

Field	Type	Description	Label
transactions	*aelf.Transaction*	Consensus system transactions.	repeated

acs4.ValidationResult

Field	Type	Description	Label
success	*bool*	Is successful.	
message	*string*	The error message.	
is_re_trigger	*bool*	Whether to trigger mining again.	

AElf.Standards.ACS6

AElf.Standards.ACS10

acs10.Dividends

Field	Type	Description	Label
value	*Dividends.ValueEntry*	The dividends, symbol -> amount.	repeated

acs10.Dividends.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs10.DonateInput

Field	Type	Description	Label
symbol	*string*	The token symbol to donate.	
amount	*int64*	The amount to donate.	

acs10.DonationReceived

Field	Type	Description	Label
from	*aelf.Address*	The address of donors.	
pool_contract	*aelf.Address*	The address of dividend pool.	
symbol	*string*	The token symbol Donated.	
amount	*int64*	The amount Donated.	

acs10.ReleaseInput

Field	Type	Description	Label
period_number	*int64*	The period number to release.	

acs10.SymbolList

Field	Type	Description	Label
value	*string*	The token symbol list.	repeated

AElf.Standards.ACS11

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.5 AElf.Contracts.Election

Election contract.

Used for voting for Block Producers.

Implement AElf Standards ACS1.

20.5.1 Contract Methods

Method Name	Request Type	Response Type
InitialElectionContract	*Election.InitialElectionContractInput*	*google.protobuf.Empty*
RegisterElectionVotingEvent	*google.protobuf.Empty*	*google.protobuf.Empty*
TakeSnapshot	*Election.TakeElectionSnapshotInput*	*google.protobuf.Empty*
AnnounceElection	*aelf.Address*	*google.protobuf.Empty*
QuitElection	*google.protobuf.StringValue*	*google.protobuf.Empty*
Vote	*Election.VoteMinerInput*	*aelf.Hash*
ChangeVotingOption	*Election.ChangeVotingOptionInput*	*google.protobuf.Empty*
Withdraw	*aelf.Hash*	*google.protobuf.Empty*
UpdateCandidateInformation	*Election.UpdateCandidateInformationInput*	*google.protobuf.Empty*
UpdateMultipleCandidateInformation	*Election.UpdateMultipleCandidateInformationInput*	*google.protobuf.Empty*
UpdateMinersCount	*Election.UpdateMinersCountInput*	*google.protobuf.Empty*
SetTreasurySchemeIds	*Election.SetTreasurySchemeIdsInput*	*google.protobuf.Empty*
SetVoteWeightInterest	*Election.VoteWeightInterestList*	*google.protobuf.Empty*
SetVoteWeightProportion	*Election.VoteWeightProportion*	*google.protobuf.Empty*
ChangeVoteWeightInterestController	*AuthorityInfo*	*google.protobuf.Empty*
ReplaceCandidatePubkey	*Election.ReplaceCandidatePubkeyInput*	*google.protobuf.Empty*
SetCandidateAdmin	*Election.SetCandidateAdminInput*	*google.protobuf.Empty*
GetCandidates	*google.protobuf.Empty*	*Election.PubkeyList*
GetVotedCandidates	*google.protobuf.Empty*	*Election.PubkeyList*
GetCandidateInformation	*google.protobuf.StringValue*	*Election.CandidateInformation*
GetVictories	*google.protobuf.Empty*	*Election.PubkeyList*
GetTermSnapshot	*Election.GetTermSnapshotInput*	*Election.TermSnapshot*
GetMinersCount	*google.protobuf.Empty*	*google.protobuf.Int32Value*
GetElectionResult	*Election.GetElectionResultInput*	*Election.ElectionResult*
GetElectorVote	*google.protobuf.StringValue*	*Election.ElectorVote*
GetElectorVoteWithRecords	*google.protobuf.StringValue*	*Election.ElectorVote*
GetElectorVoteWithAllRecords	*google.protobuf.StringValue*	*Election.ElectorVote*
GetCandidateVote	*google.protobuf.StringValue*	*Election.CandidateVote*
GetCandidateVoteWithRecords	*google.protobuf.StringValue*	*Election.CandidateVote*
GetCandidateVoteWithAllRecords	*google.protobuf.StringValue*	*Election.CandidateVote*
GetVotersCount	*google.protobuf.Empty*	*google.protobuf.Int64Value*

Method Name	Request Type	Response Type
GetVotesAmount	*google.protobuf.Empty*	*google.protobuf.Int64Value*
GetPageableCandidateInformation	*Election.PageInformation*	*Election.GetPageableCandidateInfor*
GetMinerElectionVotingItemId	*google.protobuf.Empty*	*aelf.Hash*
GetDataCenterRankingList	*google.protobuf.Empty*	*Election.DataCenterRankingList*
GetVoteWeightSetting	*google.protobuf.Empty*	*Election.VoteWeightInterestList*
GetVoteWeightProportion	*google.protobuf.Empty*	*Election.VoteWeightProportion*
GetCalculateVoteWeight	*Election.VoteInformation*	*google.protobuf.Int64Value*
GetVoteWeightInterestController	*google.protobuf.Empty*	*AuthorityInfo*
GetMinerReplacementInformation	*Election.GetMinerReplacementInformationInput*	*Election.MinerReplacementInformati*
GetCandidateAdmin	*google.protobuf.StringValue*	*aelf.Address*
GetNewestPubkey	*google.protobuf.StringValue*	*google.protobuf.StringValue*
GetReplacedPubkey	*google.protobuf.StringValue*	*google.protobuf.StringValue*

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethod-FeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethod-FeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.5.2 Contract Types

AElf.Contracts.Election

Election.CandidateDetail

Field	Type	Description	Label
candidate_information	*CandidateInformation*	The candidate information.	
obtained_votes_amount	*int64*	The number of votes a candidate has obtained.	

Election.CandidateInformation

Field	Type	Description	Label
pubkey	*string*	Candidate's public key.	
terms	*int64*	The number of terms that the candidate is elected.	repeated
produced_blocks	*int64*	The number of blocks the candidate has produced.	
missed_time_slots	*int64*	The time slot for which the candidate failed to produce blocks.	
continual_appointment_count	*int64*	The count of continual appointment.	
announcement_transaction_id	*aelf.Hash*	The transaction id when the candidate announced.	
is_current_candidate	*bool*	Indicate whether the candidate can be elected in the current term.	

Election.CandidatePubkeyReplaced

Field	Type	Description	Label
old_pubkey	*string*		
new_pubkey	*string*		

Election.CandidateVote

Field	Type	Description	Label
obtained_active_voting_record_ids	*aelf.Hash*	The active voting record ids obtained.	repeated
obtained_withdrawn_voting_record_ids	*aelf.Hash*	The active voting record ids that were withdrawn.	repeated
obtained_active_voted_votes_amount	*int64*	The total number of active votes obtained.	
all_obtained_voted_votes_amount	*int64*	The total number of votes obtained.	
obtained_active_voting_records	*ElectionVotingRecord*	The active voting records.	repeated
obtained_withdrawn_votes_records	*ElectionVotingRecord*	The voting records that were withdrawn.	repeated
pubkey	*bytes*	Public key for candidate.	

Election.ChangeVotingOptionInput

Field	Type	Description	Label
vote_id	*aelf.Hash*	The vote id to change.	
candidate_pubkey	*string*	The new candidate public key.	

Election.DataCenterRankingList

Field	Type	Description	Label
data_centers	*DataCenterRankingList.DataCentersEntry*	The top n * 5 candidates with vote amount, candidate public key -> vote amount.	repeated

Election.DataCenterRankingList.DataCentersEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Election.ElectionResult

Field	Type	Description	Label
term_number	*int64*	The term number	
results	*ElectionResult.ResultsEntry*	The election result, candidates' public key -> number of votes.	repeated
is_active	*bool*	Whether an election is currently being held.	

Election.ElectionResult.ResultsEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Election.ElectionVotingRecord

Field	Type	Description	Label
voter	*aelf.Address*	The address of voter.	
candidate	*string*	The public key of candidate.	
amount	*int64*	Amount of voting.	
term_number	*int64*	The term number of voting.	
vote_id	*aelf.Hash*	The vote id.	
lock_time	*int64*	Vote lock time.	
unlock_timestamp	*google.protobuf.Timestamp*	The unlock timestamp.	
withdraw_timestamp	*google.protobuf.Timestamp*	The withdraw timestamp.	
vote_timestamp	*google.protobuf.Timestamp*	The vote timestamp.	
is_withdrawn	*bool*	Indicates if the vote has been withdrawn.	
weight	*int64*	Vote weight for sharing bonus.	
is_change_target	*bool*	Whether vote others.	

Election.ElectorVote

Field	Type	Description	Label
active_voting_record_ids	*aelf.Hash*	The active voting record ids.	repeated
withdrawn_voting_record_ids	*aelf.Hash*	The voting record ids that were withdrawn.	repeated
active_voted_votes_amount	*int64*	The total number of active votes.	
all_voted_votes_amount	*int64*	The total number of votes (including the number of votes withdrawn).	
active_voting_records	*ElectionVotingRecord*	The active voting records.	repeated
withdrawn_votes_records	*ElectionVotingRecord*	The voting records that were withdrawn.	repeated
pubkey	*bytes*	Public key for voter.	

Election.EvilMinerDetected

Field	Type	Description	Label
pubkey	*string*	The public key of evil miner.	

Election.GetElectionResultInput

Field	Type	Description	Label
term_number	*int64*	The term number.	

Election.GetMinerReplacementInformationInput

Field	Type	Description	Label
current_miner_list	*string*	The current miner list to inspect.	repeated

Election.GetPageableCandidateInformationOutput

Field	Type	Description	Label
value	*CandidateDetail*	The details of the candidates.	repeated

Election.GetTermSnapshotInput

Field	Type	Description	Label
term_number	*int64*	The term number.	

Election.InitialElectionContractInput

Field	Type	Description	Label
minimum_lock_time	*int64*	Minimum number of seconds for locking.	
maximum_lock_time	*int64*	Maximum number of seconds for locking.	
miner_list	*string*	The current miner list.	repeated
time_each_term	*int64*	The number of seconds per term.	
miner_increase_interval	*int64*	The interval second that increases the number of miners.	

Election.MinerReplacementInformation

Field	Type	Description	Label
alternative_candidate_pubkeys	*string*	The alternative candidate public keys.	repeated
evil_miner_pubkeys	*string*	The evil miner public keys.	repeated

Election.PageInformation

Field	Type	Description	Label
start	*int32*	The start index.	
length	*int32*	The number of records.	

Election.PubkeyList

Field	Type	Description	Label
value	*bytes*	Candidates' public keys	repeated

Election.ReplaceCandidatePubkeyInput

Field	Type	Description	Label
old_pubkey	*string*		
new_pubkey	*string*		

Election.SetCandidateAdminInput

Field	Type	Description	Label
pubkey	*string*		
admin	*aelf.Address*		

Election.SetTreasurySchemeIdsInput

Field	Type	Description	Label
treasury_hash	*aelf.Hash*	The scheme id of treasury reward.	
welfare_hash	*aelf.Hash*	The scheme id of welfare reward.	
subsidy_hash	*aelf.Hash*	The scheme id of subsidy reward.	
votes_reward_hash	*aelf.Hash*	The scheme id of votes reward.	
re_election_reward_hash	*aelf.Hash*	The scheme id of re-election reward.	

Election.TakeElectionSnapshotInput

Field	Type	Description	Label
term_number	*int64*	The term number to take snapshot.	
mined_blocks	*int64*	The number of mined blocks of this term.	
round_number	*int64*	The end round number of this term.	

Election.TermSnapshot

Field	Type	Description	Label
end_round_number	*int64*	The end round number of this term.	
mined_blocks	*int64*	The number of blocks mined in this term.	
election_result	*TermSnapshot.ElectionResultEntry*	The election result, candidates' public key -> number of votes.	repeated

Election.TermSnapshot.ElectionResultEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Election.UpdateCandidateInformationInput

Field	Type	Description	Label
pubkey	*string*	The candidate public key.	
recently_produced_blocks	*int64*	The number of blocks recently produced.	
recently_missed_time_slots	*int64*	The number of time slots recently missed.	
is_evil_node	*bool*	Is it a evil node. If true will remove the candidate.	

Election.UpdateMinersCountInput

Field	Type	Description	Label
miners_count	*int32*	The count of miner.	

Election.UpdateMultipleCandidateInformationInput

Field	Type	Description	Label
value	*UpdateCandidateInformationInput*	The candidate information to update.	repeated

Election.UpdateTermNumberInput

Field	Type	Description	Label
term_number	*int64*	The term number.	

Election.VoteInformation

Field	Type	Description	Label
amount	*int64*	Amount of voting.	
lock_time	*int64*	Vote lock time.	

Election.VoteMinerInput

Field	Type	Description	Label
candidate_pubkey	*string*	The candidate public key.	
amount	*int64*	The amount token to vote.	
end_timestamp	*google.protobuf.Timestamp*	The end timestamp of this vote.	
token	*aelf.Hash*	Used to generate vote id.	

Election.VoteWeightInterest

Field	Type	Description	Label
day	*int32*	Number of days locked.	
interest	*int32*	Locked interest.	
capital	*int32*		

Election.VoteWeightInterestList

Field	Type	Description	Label
vote_weight_interest_infos	*VoteWeightInterest*	The weight of vote interest.	repeated

Election.VoteWeightProportion

Field	Type	Description	Label
time_proportion	*int32*	The weight of lock time.	
amount_proportion	*int32*	The weight of the votes cast.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.6 AElf.Contracts.Genesis

Genesis contract.

Used to manage the deployment and update of contracts.

Implement AElf Standards ACS0 and ACS1.

20.6.1 Contract Methods

Method Name	Request Type	Response Type	Description
Initialize	*Zero.InitializeInput*	*google.protobuf.Empty*	Initialize the genesis contract.
SetInitialController-Address	*aelf.Address*	*google.protobuf.Empty*	Set initial controller address for CodeCheck-Controller and ContractDeploymentController.
ChangeContractDe-ploymentController	*AuthorityInfo*	*google.protobuf.Empty*	Modify the contract deployment controller authority. Note: Only old controller has permission to do this.
ChangeCodeCheck-Controller	*AuthorityInfo*	*google.protobuf.Empty*	Modify the contract code check controller authority. Note: Only old controller has permission to do this.
GetContractDeploy-mentController	*google.protobuf.Empty*	*Authority-Info*	Query the ContractDeploymentController authority info.
GetCodeCheckCon-troller	*google.protobuf.Empty*	*Authority-Info*	Query the CodeCheckController authority info.
SetContractProposal-ExpirationTimePeriod	*Zero.SetContractProposalExpirationTimePeriodInput*	*google.protobuf.Empty*	Set the expiration time for contract proposals, 72 hours by default
GetCurrentCon-tractProposalExpira-tionTimePeriod	*google.protobuf.Empty*	*int32*	get the expiration time for the current contract proposal

AElf.Standards.ACS0

Method Name	Request Type	Response Type	Description
DeploySystemSmartContract	*acs0.SystemContractDeploymentInput*	*aelf.Address*	Deploy a system smart contract on chain and return the address of the system contract deployed.
DeploySmartContract	*acs0.ContractDeploymentInput*	*aelf.Address*	Deploy a smart contract on chain and return the address of the contract deployed.
UpdateSmartContract	*acs0.ContractUpdateInput*	*aelf.Address*	Update a smart contract on chain.
ProposeNewContract	*acs0.ContractDeploymentInput*	*aelf.Hash*	Create a proposal to deploy a new contract and returns the id of the proposed contract.
ProposeContractCodeCheck	*acs0.ContractCodeCheckInput*	*aelf.Hash*	Create a proposal to check the code of a contract and return the id of the proposed contract.
ProposeUpdateContract	*acs0.ContractUpdateInput*	*aelf.Hash*	Create a proposal to update the specified contract and return the id of the proposed contract.
ReleaseApprovedContract	*acs0.ReleaseContractInput*	*google.protobuf.Empty*	Release the contract proposal which has been approved.
ReleaseCodeCheckedContract	*acs0.ReleaseContractInput*	*google.protobuf.Empty*	Release the proposal which has passed the code check.
ValidateSystemContractAddress	*acs0.ValidateSystemContractAddressInput*	*google.protobuf.Empty*	Validate whether the input system contract exists.
SetContractProposerRequiredState	*google.protobuf.BoolValue*	*google.protobuf.Empty*	Set authority of contract deployment.
CurrentContractSerialNumber	*google.protobuf.Empty*	*google.protobuf.Int64Value*	Get the current serial number of genesis contract (corresponds to the serial number that will be given to the next deployed contract).
GetContractInfo	*aelf.Address*	*acs0.ContractInfo*	Get detailed information about the specified contract.
GetContractAuthor	*aelf.Address*	*aelf.Address*	Get author of the specified contract.
GetContractHash	*aelf.Address*	*aelf.Hash*	Get the code hash of the contract about the specified address.
GetContractAddressByName	*aelf.Hash*	*aelf.Address*	Get the address of a system contract by its name.
GetSmartContractRegistrationByAddress	*aelf.Address*	*aelf.SmartContractRegistration*	Get the registration of a smart contract by its address.
GetSmartContractRegistrationByCodeHash	*aelf.Hash*	*aelf.SmartContractRegistration*	Get the registration of a smart contract by code hash.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.6.2 Contract Types

AElf.Contracts.Genesis

Zero.ContractProposingInput

Field	Type	Description	Label
proposer	*aelf.Address*	The address of proposer for contract deployment/update.	
status	*ContractProposingInputStatus*	The status of proposal.	
expired_time	*google.protobuf.Timestamp*	The expiration time of proposal.	

Zero.InitializeInput

Field	Type	Description	Label
contract_deployment_authority_required	*bool*	Whether contract deployment/update requires authority.	

Zero.ContractProposingInputStatus

Name	Number	Description
PROPOSED	0	Proposal is proposed.
APPROVED	1	Proposal is approved by parliament.
CODE_CHECK_PROPOSED	2	Code check is proposed.
CODE_CHECKED	3	Passed code checks.

Zero.SetContractProposalExpirationTimePeriodInput

Field	Type	Description	Label
expiration_time_period	*int32*	the period of expiration time	

AElf.Standards.ACS0

acs0.CodeCheckRequired

Field	Type	Description	Label
code	*bytes*	The byte array of the contract code.	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	
category	*sint32*	The category of contract code(0: C#).	
is_system_contract	*bool*	Indicates if the contract is the system contract.	

acs0.CodeUpdated

Field	Type	Description	Label
address	*aelf.Address*	The address of the updated contract.	
old_code_hash	*aelf.Hash*	The byte array of the old contract code.	
new_code_hash	*aelf.Hash*	The byte array of the new contract code.	
version	*int32*	The version of the current contract.	

acs0.ContractCodeCheckInput

Field	Type	Description	Label
contract_input	*bytes*	The byte array of the contract code to be checked.	
is_contract_deployment	*bool*	Whether the input contract is to be deployed or updated.	
code_check_release_method	*string*	Method to call after code check complete(DeploySmartContract or UpdateSmartContract).	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	
category	*sint32*	The category of contract code(0: C#).	
is_system_contract	*bool*	Indicates if the contract is the system contract.	

acs0.ContractDeployed

Field	Type	Description	Label
author	*aelf.Address*	The author of the contract, this is the person who deployed the contract.	
code_hash	*aelf.Hash*	The hash of the contract code.	
address	*aelf.Address*	The address of the contract.	
version	*int32*	The version of the current contract.	
Name	*aelf.Hash*	The name of the contract. It has to be unique.	

acs0.ContractDeploymentInput

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	

acs0.ContractInfo

Field	Type	Description	Label
serial_number	*int64*	The serial number of the contract.	
author	*aelf.Address*	The author of the contract, this is the person who deployed the contract.	
category	*sint32*	The category of contract code(0: C#).	
code_hash	*aelf.Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

acs0.ContractProposed

Field	Type	Description	Label
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	

acs0.ContractUpdateInput

Field	Type	Description	Label
address	*aelf.Address*	The contract address that needs to be updated.	
code	*bytes*	The byte array of the new contract code.	

acs0.ReleaseContractInput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The hash of the proposal.	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	

acs0.SystemContractDeploymentInput

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
name	*aelf.Hash*	The name of the contract. It has to be unique.	
transaction_method_call_list	*SystemContractDeploymentInput.SystemTransactionMethodCallList*	An initial list of transactions for the system contract, which is executed in sequence when the contract is deployed.	

acs0.SystemContractDeploymentInput.SystemTransactionMethodCall

Field	Type	Description	Label
method_name	*string*	The method name of system transaction.	
params	*bytes*	The params of system transaction method.	

acs0.SystemContractDeploymentInput.SystemTransactionMethodCallList

Field	Type	Description	Label
value	*SystemContractDeploymentInput.SystemTransactionMethodCall*	The list of system transactions.	repeated

acs0.ValidateSystemContractAddressInput

Field	Type	Description	Label
system_contract_hash_name	*aelf.Hash*	The name hash of the contract.	
address	*aelf.Address*	The address of the contract.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.7 AElf.Contracts.MultiToken

MultiToken contract.

The MultiToken contract is mainly used to manage the user's account and transaction fees related Settings.

Implement AElf Standards ACS1 and ACS2.

20.7.1 Contract Methods

Method Name	Request Type	Response T
AdvanceResourceToken	*tokenimpl.AdvanceResourceTokenInput*	*google.proto*
TakeResourceTokenBack	*tokenimpl.TakeResourceTokenBackInput*	*google.proto*
RegisterCrossChainTokenContractAddress	*tokenimpl.RegisterCrossChainTokenContractAddressInput*	*google.proto*
SetFeeReceiver	*aelf.Address*	*google.proto*
ValidateTokenInfoExists	*tokenimpl.ValidateTokenInfoExistsInput*	*google.proto*
UpdateRental	*tokenimpl.UpdateRentalInput*	*google.proto*
UpdateRentedResources	*tokenimpl.UpdateRentedResourcesInput*	*google.proto*
TransferToContract	*tokenimpl.TransferToContractInput*	*google.proto*
ChangeSideChainRentalController	*AuthorityInfo*	*google.proto*
ChangeSymbolsToPayTXSizeFeeController	*AuthorityInfo*	*google.proto*
ChangeCrossChainTokenContractRegistrationController	*AuthorityInfo*	*google.proto*
ChangeUserFeeController	*AuthorityInfo*	*google.proto*
ChangeDeveloperController	*AuthorityInfo*	*google.proto*
GetFeeReceiver	*google.protobuf.Empty*	*aelf.Address*
GetResourceUsage	*google.protobuf.Empty*	*tokenimpl.Re*
GetSymbolsToPayTXSizeFeeController	*google.protobuf.Empty*	*AuthorityInfo*
GetCrossChainTokenContractRegistrationController	*google.protobuf.Empty*	*AuthorityInfo*
GetUserFeeController	*google.protobuf.Empty*	*tokenimpl.Us*
GetDeveloperFeeController	*google.protobuf.Empty*	*tokenimpl.De*
GetSideChainRentalControllerCreateInfo	*google.protobuf.Empty*	*AuthorityInfo*
GetVirtualAddressForLocking	*tokenimpl.GetVirtualAddressForLockingInput*	*aelf.Address*
GetOwningRental	*google.protobuf.Empty*	*tokenimpl.Ow*
GetOwningRentalUnitValue	*google.protobuf.Empty*	*tokenimpl.Ow*
Create	*token.CreateInput*	*google.proto*
Issue	*token.IssueInput*	*google.proto*
Transfer	*token.TransferInput*	*google.proto*
TransferFrom	*token.TransferFromInput*	*google.proto*
Approve	*token.ApproveInput*	*google.proto*
UnApprove	*token.UnApproveInput*	*google.proto*
Lock	*token.LockInput*	*google.proto*
Unlock	*token.UnlockInput*	*google.proto*
Burn	*token.BurnInput*	*google.proto*
ChangeTokenIssuer	*token.ChangeTokenIssuerInput*	*google.proto*
SetPrimaryTokenSymbol	*token.SetPrimaryTokenSymbolInput*	*google.proto*
CrossChainTransfer	*token.CrossChainTransferInput*	*google.proto*
CrossChainReceiveToken	*token.CrossChainReceiveTokenInput*	*google.proto*
CrossChainCreateToken	*token.CrossChainCreateTokenInput*	*google.proto*
InitializeFromParentChain	*token.InitializeFromParentChainInput*	*google.proto*
ClaimTransactionFees	*token.TotalTransactionFeesMap*	*google.proto*
ChargeTransactionFees	*token.ChargeTransactionFeesInput*	*token.Charge*
CheckThreshold	*token.CheckThresholdInput*	*google.proto*
InitialCoefficients	*google.protobuf.Empty*	*google.proto*
DonateResourceToken	*token.TotalResourceTokensMaps*	*google.proto*
ChargeResourceToken	*token.ChargeResourceTokenInput*	*google.proto*
CheckResourceToken	*google.protobuf.Empty*	*google.proto*

Method Name	Request Type	Response T
SetSymbolsToPayTxSizeFee	*token.SymbolListToPayTxSizeFee*	*google.proto*
UpdateCoefficientsForSender	*token.UpdateCoefficientsInput*	*google.proto*
UpdateCoefficientsForContract	*token.UpdateCoefficientsInput*	*google.proto*
InitializeAuthorizedController	*google.protobuf.Empty*	*google.proto*
GetTokenInfo	*token.GetTokenInfoInput*	*token.TokenI*
GetNativeTokenInfo	*google.protobuf.Empty*	*token.TokenI*
GetResourceTokenInfo	*google.protobuf.Empty*	*token.TokenI*
GetBalance	*token.GetBalanceInput*	*token.GetBa*
GetAllowance	*token.GetAllowanceInput*	*token.GetAll*
IsInWhiteList	*token.IsInWhiteListInput*	*google.proto*
GetLockedAmount	*token.GetLockedAmountInput*	*token.GetLo*
GetCrossChainTransferTokenContractAddress	*token.GetCrossChainTransferTokenContractAddressInput*	*aelf.Address*
GetPrimaryTokenSymbol	*google.protobuf.Empty*	*google.proto*
GetCalculateFeeCoefficientsForContract	*google.protobuf.Int32Value*	*token.Calcul*
GetCalculateFeeCoefficientsForSender	*google.protobuf.Empty*	*token.Calcul*
GetSymbolsToPayTxSizeFee	*google.protobuf.Empty*	*token.Symbo*
GetLatestTotalTransactionFeesMapHash	*google.protobuf.Empty*	*aelf.Hash*
GetLatestTotalResourceTokensMapsHash	*google.protobuf.Empty*	*aelf.Hash*
IsTokenAvailableForMethodFee	*google.protobuf.StringValue*	*google.proto*
ConfigMethodFeeFreeAllowances	*token.MethodFeeFreeAllowancesConfig*	*google.proto*
SetTransactionFeeDelegations	*token.SetTransactionFeeDelegationsInput*	*token.SetTra*
RemoveTransactionFeeDelegator	*token.RemoveTransactionFeeDelegatorInput*	*google.proto*
RemoveTransactionFeeDelegatee	*token.RemoveTransactionFeeDelegateeInput*	*google.proto*
GetMethodFeeFreeAllowances	*aelf.Address*	*token.Metho*
GetMethodFeeFreeAllowancesConfig	*google.protobuf.Empty*	*token.Metho*
GetTransactionFeeDelegationsOfADelegatee	*token.GetTransactionFeeDelegationsOfADelegateeInput*	*token.Transa*

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethod-FeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethod-FeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS2

Method Name	Request Type	Response Type	Description
GetResource-Info	*aelf.Transaction*	*acs2.ResourceInfo*	Gets the resource information that the transaction execution depends on.

20.7.2 Contract Types

AElf.Contracts.MultiToken

tokenimpl.AdvanceResourceTokenInput

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address to transfer.	
resource_token_symbol	*string*	The resource token symbol to transfer.	
amount	*int64*	The amount of resource token to transfer.	

tokenimpl.DeveloperFeeController

Field	Type	Description	Label
root_controller	*AuthorityInfo*	The association that governs the organization.	
parliament_controller	*AuthorityInfo*	The parliament organization of members.	
developer_controller	*AuthorityInfo*	The developer organization of members.	

tokenimpl.GetVirtualAddressForLockingInput

Field	Type	Description	Label
address	*aelf.Address*	The address of the lock.	
lock_id	*aelf.Hash*	The id of the lock.	

tokenimpl.OwningRental

Field	Type	Description	Label
resource_amount	*OwningRental.ResourceAmountEntry*	The amount of resource tokens owed, symbol -> amount.	repeated

tokenimpl.OwningRental.ResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

tokenimpl.OwningRentalUnitValue

Field	Type	Description	Label
resource_unit_value	*OwningRentalUnitValue.ResourceUnitValueEntry*	Resource unit price, symbol -> unit price.	repeated

tokenimpl.OwningRentalUnitValue.ResourceUnitValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

tokenimpl.RegisterCrossChainTokenContractAddressInput

Field	Type	Description	Label
from_chain_id	*int32*	The source chain id.	
parent_chain_height	*int64*	The parent chain height of the transaction.	
transaction_bytes	*bytes*	The raw bytes of the transfer transaction.	
merkle_path	*aelf.MerklePath*	The merkle path created from the transaction.	
token_contract_address	*aelf.Address*	The token contract address.	

tokenimpl.ResourceUsage

Field	Type	Description	Label
value	*ResourceUsage.ValueEntry*	The amount of resource tokens usage, symbol -> amount.	repeated

tokenimpl.ResourceUsage.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

tokenimpl.TakeResourceTokenBackInput

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address to take back.	
resource_token_symbol	*string*	The resource token symbol to take back.	
amount	*int64*	The amount of resource token to take back.	

tokenimpl.TransferToContractInput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
amount	*int64*	The amount of token.	
memo	*string*	The memo.	

tokenimpl.UpdateRentalInput

Field	Type	Description	Label
rental	*UpdateRentalInput.RentalEntry*	The unit price of resource tokens, symbol -> unit price.	repeated

tokenimpl.UpdateRentalInput.RentalEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

tokenimpl.UpdateRentedResourcesInput

Field	Type	Description	Label
resource_amount	*UpdateRentedResourcesInput.ResourceAmountEntry*	Amount of resource tokens consumed per minute, symbol -> resource consumption.	repeated

tokenimpl.UpdateRentedResourcesInput.ResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

tokenimpl.UserFeeController

Field	Type	Description	Label
root_controller	*AuthorityInfo*	The association that governs the organization.	
parliament_controller	*AuthorityInfo*	The parliament organization of members.	
referendum_controller	*AuthorityInfo*	The referendum organization of members.	

tokenimpl.ValidateTokenInfoExistsInput

Field	Type	Description	Label
symbol	*string*	The symbol of the token.	
token_name	*string*	The full name of the token.	
total_supply	*int64*	The total supply of the token.	
decimals	*int32*	The precision of the token.	
issuer	*aelf.Address*	The address that created the token.	
is_burnable	*bool*	A flag indicating if this token is burnable.	
issue_chain_id	*int32*	The chain id of the token.	

token.AllCalculateFeeCoefficients

Field	Type	Description	Label
value	*CalculateFeeCoefficients*	The coefficients of fee Calculation.	repeated

token.ApproveInput

Field	Type	Description	Label
spender	*aelf.Address*	The address that allowance will be increased.	
symbol	*string*	The symbol of token to approve.	
amount	*int64*	The amount of token to approve.	

token.Approved

Field	Type	Description	Label
owner	*aelf.Address*	The address of the token owner.	
spender	*aelf.Address*	The address that allowance be increased.	
symbol	*string*	The symbol of approved token.	
amount	*int64*	The amount of approved token.	

token.BurnInput

Field	Type	Description	Label
symbol	*string*	The symbol of token to burn.	
amount	*int64*	The amount of token to burn.	

token.Burned

Field	Type	Description	Label
burner	*aelf.Address*	The address who wants to burn token.	
symbol	*string*	The symbol of burned token.	
amount	*int64*	The amount of burned token.	

token.CalculateFeeAlgorithmUpdated

Field	Type	Description	Label
all_type_fee_coefficients	*AllCalculateFeeCoefficients*	All calculate fee coefficients after modification.	

token.CalculateFeeCoefficients

Field	Type	Description	Label
fee_token_type	*int32*	The resource fee type, like READ, WRITE, etc.	
piece_coefficients_list	*CalculateFeePieceCoefficients*	Coefficients of one single piece.	repeated

token.CalculateFeePieceCoefficients

Field	Type	Description	Label
value	*int32*	Coefficients of one single piece. The first char is its type: liner / power. The second char is its piece upper bound.	repeated

token.ChainPrimaryTokenSymbolSet

Field	Type	Description	Label
token_symbol	*string*	The symbol of token.	

token.ChangeTokenIssuerInput

Field	Type	Description	Label
symbol	*string*	The token symbol.	
new_token_Issuer	*aelf.Address*	The new token issuer for change.	

token.ChargeResourceTokenInput

Field	Type	Description	Label
cost_dic	*ChargeResourceTokenInput.CostDicEntry*	Collection of charge resource token, Symbol->Amount.	repeated
caller	*aelf.Address*	The sender of the transaction.	

token.ChargeResourceTokenInput.CostDicEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

token.ChargeTransactionFeesInput

Field	Type	Description	Label
method_name	*string*	The method name of transaction.	
contract_address	*aelf.Address*	The contract address of transaction.	
transaction_size_fee	*int64*	The amount of transaction size fee.	
symbols_to_pay_tx_size_fee	*SymbolToPayTxSizeFee*	Transaction fee token information.	repeated

token.ChargeTransactionFeesOutput

Field	Type	Description	Label
success	*bool*	Whether the charge was successful.	
charging_information	*string*	The charging information.	

token.CheckThresholdInput

Field	Type	Description	Label
sender	*aelf.Address*	The sender of the transaction.	
symbol_to_threshold	*CheckThresholdInput.SymbolToThresholdEntry*	The threshold to set, Symbol->Threshold.	repeated
is_check_allowance	*bool*	Whether to check the allowance.	

token.CheckThresholdInput.SymbolToThresholdEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

token.ContractTotalResourceTokens

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address.	
tokens_map	*TotalResourceTokensMap*	Resource tokens to charge.	

token.CreateInput

Field	Type	Description	Label
symbol	*string*	The symbol of the token.	
token_name	*string*	The full name of the token.	
total_supply	*int64*	The total supply of the token.	
decimals	*int32*	The precision of the token	
issuer	*aelf.Address*	The address that created the token.	
is_burnable	*bool*	A flag indicating if this token is burnable.	
lock_white_list	*aelf.Address*	A whitelist address list used to lock tokens.	repeated
issue_chain_id	*int32*	The chain id of the token.	

token.CrossChainCreateTokenInput

Field	Type	Description	Label
from_chain_id	*int32*	The chain id of the chain on which the token was created.	
parent_chain_height	*int64*	The height of the transaction that created the token.	
transaction_bytes	*bytes*	The transaction that created the token.	
merkle_path	*aelf.MerklePath*	The merkle path created from the transaction that created the transaction.	

token.CrossChainReceiveTokenInput

Field	Type	Description	Label
from_chain_id	*int32*	The source chain id.	
parent_chain_height	*int64*	The height of the transfer transaction.	
transfer_transaction_bytes	*bytes*	The raw bytes of the transfer transaction.	
merkle_path	*aelf.MerklePath*	The merkle path created from the transfer transaction.	

token.CrossChainReceived

Field	Type	Description	Label
from	*aelf.Address*	The source address of the transferred token.	
to	*aelf.Address*	The destination address of the transferred token.	
symbol	*string*	The symbol of the received token.	
amount	*int64*	The amount of the received token.	
memo	*string*	The memo.	
from_chain_id	*int32*	The destination chain id.	
issue_chain_id	*int32*	The chain id of the token.	
parent_chain_height	*int64*	The parent chain height of the transfer transaction.	

token.CrossChainTransferInput

Field	Type	Description	Label
to	*aelf.Address*	The receiver of transfer.	
symbol	*string*	The symbol of token.	
amount	*int64*	The amount of token to transfer.	
memo	*string*	The memo.	
to_chain_id	*int32*	The destination chain id.	
issue_chain_id	*int32*	The chain id of the token.	

token.CrossChainTransferred

Field	Type	Description	Label
from	*aelf.Address*	The source address of the transferred token.	
to	*aelf.Address*	The destination address of the transferred token.	
symbol	*string*	The symbol of the transferred token.	
amount	*int64*	The amount of the transferred token.	
memo	*string*	The memo.	
to_chain_id	*int32*	The destination chain id.	
issue_chain_id	*int32*	The chain id of the token.	

token.ExtraTokenListModified

Field	Type	Description	Label
symbol_list_to_pay_tx_size_fee	*SymbolListToPayTxSizeFee*	Transaction fee token information.	

token.GetAllowanceInput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
owner	*aelf.Address*	The address of the token owner.	
spender	*aelf.Address*	The address of the spender.	

token.GetAllowanceOutput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
owner	*aelf.Address*	The address of the token owner.	
spender	*aelf.Address*	The address of the spender.	
allowance	*int64*	The amount of allowance.	

token.GetBalanceInput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
owner	*aelf.Address*	The target address of the query.	

token.GetBalanceOutput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
owner	*aelf.Address*	The target address of the query.	
balance	*int64*	The balance of the owner.	

token.GetCrossChainTransferTokenContractAddressInput

Field	Type	Description	Label
chainId	*int32*	The chain id.	

token.GetLockedAmountInput

Field	Type	Description	Label
address	*aelf.Address*	The address of the lock.	
symbol	*string*	The token symbol.	
lock_id	*aelf.Hash*	The id of the lock.	

token.GetLockedAmountOutput

Field	Type	Description	Label
address	*aelf.Address*	The address of the lock.	
symbol	*string*	The token symbol.	
lock_id	*aelf.Hash*	The id of the lock.	
amount	*int64*	The locked amount.	

token.GetTokenInfoInput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	

token.InitializeFromParentChainInput

Field	Type	Description	Label
resource_amount	*InitializeFromParentChainInput.ResourceAmountEntry*	The amount of resource.	repeated
registered_other_token_contract_addresses	*InitializeFromParentChainInput.RegisteredOtherTokenContractAddressesEntry*	The token contract addresses.	repeated
creator	*aelf.Address*	The creator the side chain.	

token.InitializeFromParentChainInput.RegisteredOtherTokenContractAddressesEntry

Field	Type	Description	Label
key	*int32*		
value	*aelf.Address*		

token.InitializeFromParentChainInput.ResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

token.IsInWhiteListInput

Field	Type	Description	Label
symbol	*string*	The symbol of token.	
address	*aelf.Address*	The address to check.	

token.IssueInput

Field	Type	Description	Label
symbol	*string*	The token symbol to issue.	
amount	*int64*	The token amount to issue.	
memo	*string*	The memo.	
to	*aelf.Address*	The target address to issue.	

token.Issued

Field	Type	Description	Label
symbol	*string*	The symbol of issued token.	
amount	*int64*	The amount of issued token.	
memo	*string*	The memo.	
to	*aelf.Address*	The issued target address.	

token.LockInput

Field	Type	Description	Label
address	*aelf.Address*	The one want to lock his token.	
lock_id	*aelf.Hash*	Id of the lock.	
symbol	*string*	The symbol of the token to lock.	
usage	*string*	a memo.	
amount	*int64*	The amount of tokens to lock.	

token.RentalAccountBalanceInsufficient

Field	Type	Description	Label
symbol	*string*	The symbol of insufficient rental account balance.	
amount	*int64*	The balance of the account.	

token.RentalCharged

Field	Type	Description	Label
symbol	*string*	The symbol of rental fee charged.	
amount	*int64*	The amount of rental fee charged.	

token.SetPrimaryTokenSymbolInput

Field	Type	Description	Label
symbol	*string*	The symbol of the token.	

token.SymbolListToPayTxSizeFee

Field	Type	Description	Label
symbols_to_pay_tx_size_fee	*SymbolToPayTxSizeFee*	Transaction fee token information.	repeated

token.SymbolToPayTxSizeFee

Field	Type	Description	Label
token_symbol	*string*	The symbol of token.	
base_token_weight	*int32*	The charge weight of primary token.	
added_token_weight	*int32*	The new added token charge weight. For example, the charge weight of primary Token is set to 1. The newly added token charge weight is set to 10. If the transaction requires 1 unit of primary token, the user can also pay for 10 newly added tokens.	

token.TokenCreated

Field	Type	Description	Label
symbol	*string*	The symbol of the token.	
token_name	*string*	The full name of the token.	
total_supply	*int64*	The total supply of the token.	
decimals	*int32*	The precision of the token.	
issuer	*aelf.Address*	The address that created the token.	
is_burnable	*bool*	A flag indicating if this token is burnable.	
issue_chain_id	*int32*	The chain id of the token.	

token.TokenInfo

Field	Type	Description	Label
symbol	*string*	The symbol of the token.f	
token_name	*string*	The full name of the token.	
supply	*int64*	The current supply of the token.	
total_supply	*int64*	The total supply of the token.	
decimals	*int32*	The precision of the token.	
issuer	*aelf.Address*	The address that created the token.	
is_burnable	*bool*	A flag indicating if this token is burnable.	
issue_chain_id	*int32*	The chain id of the token.	
issued	*int64*	The amount of issued tokens.	

token.TokenInfoList

Field	Type	Description	Label
value	*TokenInfo*	List of token information.	repeated

token.TotalResourceTokensMap

Field	Type	Description	Label
value	*TotalResourceTokensMap.ValueEntry*	Resource token dictionary, Symbol->Amount.	repeated

token.TotalResourceTokensMap.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

token.TotalResourceTokensMaps

Field	Type	Description	Label
value	*ContractTotalResourceTokens*	Resource tokens to charge.	repeated
block_hash	*aelf.Hash*	The hash of the block processing the transaction.	
block_height	*int64*	The height of the block processing the transaction.	

token.TotalTransactionFeesMap

Field	Type	Description	Label
value	*TotalTransaction-FeesMap.ValueEntry*	Token dictionary that charge transaction fee, Symbol->Amount.	re-peated
block_hash	*aelf.Hash*	The hash of the block processing the transaction.	
block_height	*int64*	The height of the block processing the transaction.	

token.TotalTransactionFeesMap.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

token.TransactionFeeBill

Field	Type	Description	Label
fees_map	*TransactionFeeBill.FeesMapEntry*	The transaction fee dictionary, Symbol->fee.	repeated

token.TransactionFeeBill.FeesMapEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

token.TransferFromInput

Field	Type	Description	Label
from	*aelf.Address*	The source address of the token.	
to	*aelf.Address*	The destination address of the token.	
symbol	*string*	The symbol of the token to transfer.	
amount	*int64*	The amount to transfer.	
memo	*string*	The memo.	

token.TransferInput

Field	Type	Description	Label
to	*aelf.Address*	The receiver of the token.	
symbol	*string*	The token symbol to transfer.	
amount	*int64*	The amount to to transfer.	
memo	*string*	The memo.	

token.Transferred

Field	Type	Description	Label
from	*aelf.Address*	The source address of the transferred token.	
to	*aelf.Address*	The destination address of the transferred token.	
symbol	*string*	The symbol of the transferred token.	
amount	*int64*	The amount of the transferred token.	
memo	*string*	The memo.	

token.UnApproveInput

Field	Type	Description	Label
spender	*aelf.Address*	The address that allowance will be decreased.	
symbol	*string*	The symbol of token to un-approve.	
amount	*int64*	The amount of token to un-approve.	

token.UnApproved

Field	Type	Description	Label
owner	*aelf.Address*	The address of the token owner.	
spender	*aelf.Address*	The address that allowance be decreased.	
symbol	*string*	The symbol of un-approved token.	
amount	*int64*	The amount of un-approved token.	

token.UnlockInput

Field	Type	Description	Label
address	*aelf.Address*	The one want to un-lock his token.	
lock_id	*aelf.Hash*	Id of the lock.	
symbol	*string*	The symbol of the token to un-lock.	
usage	*string*	a memo.	
amount	*int64*	The amount of tokens to un-lock.	

token.UpdateCoefficientsInput

Field	Type	Description	Label
piece_numbers	*int32*	The specify pieces gonna update.	repeated
coefficients	*CalculateFeeCoefficients*	Coefficients of one single type.	

token.FeeTypeEnum

Name	Number	Description
READ	0	
STORAGE	1	
WRITE	2	
TRAFFIC	3	
TX	4	

token.MethodFeeFreeAllowancesConfig

Field	Type	Description	Label
free_allowances	*token.MethodFeeFreeAllowances*	The allowance of each token when a user gets his allowance of the full amount.	
refresh_seconds	*int64*	The time needed for a user's allowance to be refreshed back to the full amount. Unit: second	
threshold	*int64*	The required amount of ELF in possession for a user to be eligible for transaction fee exemption.	

token.SetTransactionFeeDelegationsInput

Field	Type	Description	Label
delegator_address	*aelf.Addresss*	The address of delegator.	
delegations	*map<string, int64>*	<token symbol, delegation>	

token.SetTransactionFeeDelegationsOutput

Field	Type	Description	Label
success	*bool*	Whether set delegation success.	

token.RemoveTransactionFeeDelegatorInput

Field	Type	Description	Label
delegator_address	*aelf.Addresss*	The address of delegator	

token.RemoveTransactionFeeDelegateeInput

Field	Type	Description	Label
delegatee_address	*aelf.Addresss*	The address of delegatee	

token.MethodFeeFreeAllowances

Field	Type	Description	Label
value	*token.MethodFeeFreeAllowance*		repeated

token.MethodFeeFreeAllowance

Field	Type	Description	Label
symbol	*string*	Token symbol	
amount	*int64*	The amount of fee free allowance	

token.GetTransactionFeeDelegationsOfADelegateeInput

Field	Type	Description	Label
delegatee_address	*aelf.Addresss*	The address of delegatee	
delegator_address	*aelf.Addresss*	The address of delegator	

token.TransactionFeeDelegations

Field	Type	Description	Label
delegations	*map<string, int64>*	The number of tokens allowed to be delegated	
block_height	*int64*	The block height when the information of delegation is added	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS2

acs2.ResourceInfo

Field	Type	Description	Label
write_paths	*aelf.ScopedStatePath*	The state path that depends on when writing.	repeated
read_paths	*aelf.ScopedStatePath*	The state path that depends on when reading.	repeated
non_parallelizable	*bool*	Whether the transaction is not executed in parallel.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.8 AElf.Contracts.Profit

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.

Implement AElf Standards ACS1.

20.8.1 Contract Methods

Method Name	Request Type	Response Type	Description
CreateScheme	*Profit.CreateSchemeInput*	*aelf.Hash*	Create a scheme for profit distribution, and return the created scheme id.
AddBeneficiary	*Profit.AddBeneficiaryInput*	*google.protobuf.Empty*	Add beneficiary to scheme.
RemoveBeneficiary	*Profit.RemoveBeneficiaryInput*	*google.protobuf.Empty*	Remove beneficiary from scheme.
AddBeneficiaries	*Profit.AddBeneficiariesInput*	*google.protobuf.Empty*	Batch add beneficiary to scheme.
RemoveBeneficiaries	*Profit.RemoveBeneficiariesInput*	*google.protobuf.Empty*	Batch remove beneficiary from scheme.
ContributeProfits	*Profit.ContributeProfitsInput*	*google.protobuf.Empty*	Contribute profit to a scheme.
ClaimProfits	*Profit.ClaimProfitsInput*	*google.protobuf.Empty*	The beneficiary draws tokens from the scheme.
DistributeProfits	*Profit.DistributeProfitsInput*	*google.protobuf.Empty*	Distribute profits to schemes, including its sub scheme according to period and token symbol, should be called by the manager.
AddSubScheme	*Profit.AddSubSchemeInput*	*google.protobuf.Empty*	Add sub scheme to a scheme. This will effectively add the specified sub-scheme as a beneficiary of the parent scheme.
RemoveSubScheme	*Profit.RemoveSubSchemeInput*	*google.protobuf.Empty*	Remove sub scheme from a scheme.
ResetManager	*Profit.ResetManagerInput*	*google.protobuf.Empty*	Reset the manager of a scheme.
GetManagingSchemeIds	*Profit.GetManagingSchemeIdsInput*	*Profit.CreatedSchemeIds*	Get all schemes managed by the specified manager.
GetScheme	*aelf.Hash*	*Profit.Scheme*	Get scheme according to scheme id.
GetSchemeAddress	*Profit.SchemePeriod*	*aelf.Address*	Get the virtual address of the number of period of the scheme.
GetDistributedProfitsInfo	*Profit.SchemePeriod*	*Profit.DistributedProfitsInfo*	Query the distributed profit information for the specified period.
GetProfitDetails	*Profit.GetProfitDetailsInput*	*Profit.ProfitDetails*	Query the beneficiary's profit information on the scheme.
GetProfitAmount	*Profit.GetProfitAmountInput*	*google.protobuf.Int64Value*	Query the amount of profit according to token symbol. (up to 10 periods).
GetProfitsMap	*Profit.ClaimProfitsInput*	*Profit.ReceivedProfitsMap*	Query all profit (up to 10 periods).

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.8.2 Contract Types

AElf.Contracts.Profit

Profit.AddBeneficiariesInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
beneficiary_shares	*BeneficiaryShare*	The beneficiary information.	repeated
end_period	*int64*	The end period which the beneficiary receives the profit.	

Profit.AddBeneficiaryInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
beneficiary_share	*BeneficiaryShare*	The beneficiary information.	
end_period	*int64*	The end period which the beneficiary receives the profit.	

Profit.AddSubSchemeInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The parent scheme id.	
sub_scheme_id	*aelf.Hash*	The sub scheme id.	
sub_scheme_shares	*int64*	The profit weight of sub scheme.	

Profit.BeneficiaryShare

Field	Type	Description	Label
beneficiary	*aelf.Address*	The address of beneficiary.	
shares	*int64*	The profit weight of the beneficiary in the scheme.	

Profit.ClaimProfitsInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
beneficiary	*aelf.Address*	The address of beneficiary.	

Profit.ContributeProfitsInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id to contribute.	
amount	*int64*	The amount to contribute.	
period	*int64*	The number of periods in which the income is used for dividends.	
symbol	*string*	The token symbol to contribute.	

Profit.CreateSchemeInput

Field	Type	Description	Label
profit_receiving_due_period_count	*int64*	Period of profit distribution.	
is_release_all_balance_every_time_by_default	*bool*	Whether all the schemes balance will be distributed during distribution each period.	
delay_distribute_period_count	*int32*	Delay distribute period.	
manager	*aelf.Address*	The manager of this scheme, the default is the creator.	
can_remove_beneficiary_directly	*bool*	Whether you can directly remove the beneficiary.	
token	*aelf.Hash*	Use to generate scheme id.	

Profit.CreatedSchemeIds

Field	Type	Description	Label
scheme_ids	*aelf.Hash*	The scheme ids.	repeated

Profit.DistributeProfitsInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id to distribute.	
period	*int64*	The period number to distribute, should be the current period.	
amounts_map	*DistributeProfitsInput.AmountsMapEntry*	The amount to distribute, symbol -> amount.	repeated

Profit.DistributeProfitsInput.AmountsMapEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Profit.DistributedProfitsInfo

Field	Type	Description	Label
total_shares	*int64*	The total amount of shares in this scheme at the current period.	
amounts_map	*DistributedProfitsInfo.AmountsMapEntry*	The contributed amount in this scheme at the current period.	repeated
is_released	*bool*	Whether released.	

Profit.DistributedProfitsInfo.AmountsMapEntry

Field	Type	Description	Label
key	*string*		
value	*sint64*		

Profit.GetManagingSchemeIdsInput

Field	Type	Description	Label
manager	*aelf.Address*	The manager address.	

Profit.GetProfitAmountInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
symbol	*string*	The token symbol.	
beneficiary	*aelf.Address*	The beneficiary's address.	

Profit.GetProfitDetailsInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
beneficiary	*aelf.Address*	The address of beneficiary.	

Profit.ProfitDetail

Field	Type	Description	Label
start_period	*int64*	The start period number.	
end_period	*int64*	The end period number.	
shares	*int64*	The weight of the proceeds on the current period of the scheme.	
last_profit_period	*int64*	The last period number that the beneficiary received the profit.	
is_weight_removed	*bool*	Whether the weight has been removed.	

Profit.ProfitDetails

Field	Type	Description	Label
details	*ProfitDetail*	The profit information.	repeated

Profit.ProfitsClaimed

Field	Type	Description	Label
beneficiary	*aelf.Address*	The beneficiary's address claimed.	
symbol	*string*	The token symbol claimed.	
amount	*int64*	The amount claimed.	
period	*int64*	The period number claimed.	
claimer_shares	*int64*	The shares of the claimer.	
total_shares	*int64*	The total shares at the current period.	

Profit.ReceivedProfitsMap

Field	Type	Description	Label
value	*ReceivedProfitsMap.ValueEntry*	The collection of profits received, token symbol -> amount.	repeated

Profit.ReceivedProfitsMap.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Profit.RemoveBeneficiariesInput

Field	Type	Description	Label
beneficiaries	*aelf.Address*	The addresses of beneficiary.	repeated
scheme_id	*aelf.Hash*	The scheme id.	

Profit.RemoveBeneficiaryInput

Field	Type	Description	Label
beneficiary	*aelf.Address*	The address of beneficiary.	
scheme_id	*aelf.Hash*	The scheme id.	

Profit.RemoveSubSchemeInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The parent scheme id.	
sub_scheme_id	*aelf.Hash*	The sub scheme id.	

Profit.ResetManagerInput

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
new_manager	*aelf.Address*	The address of new manager.	

Profit.Scheme

Field	Type	Description	Label
virtual_address	*aelf.Address*	The virtual address of the scheme.	
total_shares	*int64*	The total weight of the scheme.	
manager	*aelf.Address*	The manager of the scheme.	
current_period	*int64*	The current period.	
sub_schemes	*SchemeBeneficiaryShare*	Sub schemes information.	repeated
can_remove_beneficiary_directly	*bool*	Whether you can directly remove the beneficiary.	
profit_receiving_due_period_count	*int64*	Period of profit distribution.	
is_release_all_balance_every_time_by_default	*bool*	Whether all the schemes balance will be distributed during distribution each period.	
scheme_id	*aelf.Hash*	The is of the scheme.	
delay_distribute_period_count	*int32*	Delay distribute period.	
cached_delay_total_shares	*Scheme.CachedDelayTotalSharesEntry*	Record the scheme's current total share for deferred distribution of benefits, period -> total shares.	repeated
received_token_symbols	*string*	The received token symbols.	repeated

Profit.Scheme.CachedDelayTotalSharesEntry

Field	Type	Description	Label
key	*int64*		
value	*int64*		

Profit.SchemeBeneficiaryShare

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The id of the sub scheme.	
shares	*int64*	The weight of the sub scheme.	

Profit.SchemeCreated

Field	Type	Description	Label
virtual_address	*aelf.Address*	The virtual address of the created scheme.	
manager	*aelf.Address*	The manager of the created scheme.	
profit_receiving_due_period_count	*int64*	Period of profit distribution.	
is_release_all_balance_every_time_by_default	*bool*	Whether all the schemes balance will be distributed during distribution each period.	
scheme_id	*aelf.Hash*	The id of the created scheme.	

Profit.SchemePeriod

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
period	*int64*	The period number.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.9 AElf.Contracts.CrossChain

Cross-Chain contract.

Implement AElf Standards ACS1 and ACS7.

20.9.1 Contract Methods

Method Name	Request Type	Response Type	Description
Initialize	*CrossChain.InitializeInput*	*google.protobuf.Empty*	Propose once cross chain indexing.
SetInitialSideChainLifetimeControllerAddress	*aelf.Address*	*google.protobuf.Empty*	Set the initial SideChainLifetimeController address which should be parliament organization by default.
SetInitialIndexingControllerAddress	*aelf.Address*	*google.protobuf.Empty*	Set the initial CrossChainIndexingController address which should be parliament organization by default.
ChangeCrossChainIndexingController	*AuthorityInfo*	*google.protobuf.Empty*	Change the cross chain indexing controller.
ChangeSideChainLifetimeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the lifetime controller of the side chain.
ChangeSideChainIndexingFeeController	*CrossChain.ChangeSideChainIndexingFeeControllerInput*	*google.protobuf.Empty*	Change indexing fee adjustment controller for specific side chain.
AcceptCrossChainIndexingProposal	*CrossChain.AcceptCrossChainIndexingProposalInput*	*google.protobuf.Empty*	When the indexing proposal is released, clean up the pending proposal.
GetSideChainCreator	*google.protobuf.Int32Value*	*aelf.Address*	Get the side chain creator address according to side chain id.
GetChainStatus	*google.protobuf.Int32Value*	*CrossChain.GetChainStatusOutput*	Get the current status of side chain according to side chain id.
GetSideChainHeight	*google.protobuf.Int32Value*	*google.protobuf.Int64Value*	Get the side chain height according to side chain id.
GetParentChainHeight	*google.protobuf.Empty*	*google.protobuf.Int64Value*	Get the height of parent chain.
GetParentChainId	*google.protobuf.Empty*	*google.protobuf.Int32Value*	Get the chain id of parent chain.
GetSideChainBalance	*google.protobuf.Int32Value*	*google.protobuf.Int64Value*	Get the balance of side chain indexing according to side chain id.
GetSideChainIndexingFeeDebt	*google.protobuf.Int32Value*	*google.protobuf.Int64Value*	Get the fee debt of side chain indexing according to side chain id.
GetIndexingProposalStatus	*google.protobuf.Empty*	*CrossChain.GetIndexingProposalStatusOutput*	Get the status of the current indexing proposal.
GetSideChainIndexingFeePrice	*google.protobuf.Int32Value*	*google.protobuf.Int64Value*	Get the side chain indexing fee price according to side chain id.
GetSideChainLifetimeController	*google.protobuf.Empty*	*AuthorityInfo*	Get the lifetime controller of the side chain.
GetCrossChainIndexingController	*google.protobuf.Empty*	*AuthorityInfo*	Get the cross chain indexing controller.
GetSideChainIndexingFeeController	*google.protobuf.Int32Value*	*AuthorityInfo*	Get the indexing fee controller of side chain according to side chain id.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS7

Method Name	Request Type	Response Type	Description
ProposeCrossChainIndexing	*acs7.CrossChainBlockData*	*google.protobuf.Empty*	Propose once cross chain indexing.
ReleaseCrossChainIndexingProposal	*acs7.ReleaseCrossChainIndexingProposalInput*	*google.protobuf.Empty*	Release the proposed indexing if already approved.
RequestSideChainCreation	*acs7.SideChainCreationRequest*	*google.protobuf.Empty*	Request side chain creation.
ReleaseSideChainCreation	*acs7.ReleaseSideChainCreationInput*	*google.protobuf.Empty*	Release the side chain creation request if already approved and it will call the method CreateSideChain.
CreateSideChain	*acs7.CreateSideChainInput*	*google.protobuf.Int32Value*	Create the side chain and returns the newly created side chain ID. Only SideChainLifetimeController is permitted to invoke this method.
Recharge	*acs7.RechargeInput*	*google.protobuf.Empty*	Recharge for the specified side chain.
DisposeSideChain	*google.protobuf.Int32Value*	*google.protobuf.Int32Value*	Dispose a side chain according to side chain id. Only SideChainLifetimeController is permitted to invoke this method.
AdjustIndexingFeePrice	*acs7.AdjustIndexingFeeInput*	*google.protobuf.Empty*	Adjust side chain indexing fee. Only IndexingFeeController is permitted to invoke this method.
VerifyTransaction	*acs7.VerifyTransactionInput*	*google.protobuf.BoolValue*	Verify cross chain transaction.
GetSideChainIdAndHeight	*google.protobuf.Empty*	*acs7.ChainIdAndHeightDict*	Gets all the side chain id and height of the current chain.
GetSideChainIndexingInformationList	*google.protobuf.Empty*	*acs7.SideChainIndexingInformationList*	Get indexing information of side chains.
GetAllChainsIdAndHeight	*google.protobuf.Empty*	*acs7.ChainIdAndHeightDict*	Get id and recorded height of all chains.
GetIndexedSideChainBlockDataByHeight	*google.protobuf.Int64Value*	*acs7.IndexedSideChainBlockData*	Get block data of indexed side chain according to height.
GetBoundParentChainHeightAndMerklePathByHeight	*google.protobuf.Int64Value*	*acs7.CrossChainMerkleProofContext*	Get merkle path bound up with side chain according to height.
GetChainInitializationData	*google.protobuf.Int32Value*	*acs7.ChainInitializationData*	Get initialization data for specified side chain.

20.9.2 Contract Types

AElf.Contracts.CrossChain

CrossChain.AcceptCrossChainIndexingProposalInput

Field	Type	Description	Label
chain_id	*int32*	The chain id of accepted indexing.	

CrossChain.ChainIndexingProposal

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of cross chain indexing proposal.	
proposer	*aelf.Address*	The proposer of cross chain indexing.	
proposed_cross_chain_block_data	*acs7.CrossChainBlockData*	The cross chain data proposed.	
status	*CrossChainIndexingProposalStatus*	The status of of cross chain indexing proposal.	
chain_id	*int32*	The chain id of the indexing.	

CrossChain.ChangeSideChainIndexingFeeControllerInput

Field	Type	Description	Label
chain_id	*int32*	The side chain id.	
authority_info	*AuthorityInfo*	The changed controller of indexing fee.	

CrossChain.CrossChainIndexingControllerChanged

Field	Type	Description	Label
authority_info	*AuthorityInfo*	The changed controller of indexing.	

CrossChain.Disposed

Field	Type	Description	Label
chain_id	*int32*	The disposed side chain id.	

CrossChain.GetChainStatusOutput

Field	Type	Description	Label
status	*SideChainStatus*	The status of side chain.	

CrossChain.GetIndexingProposalStatusOutput

Field	Type	Description	Label
chain_indexing_proposal_status	*GetIndexingProposalStatusOutput.ChainIndexingProposalStatusEntry*	The collection of pending indexing proposal, the key is chain id.	repeated

CrossChain.GetIndexingProposalStatusOutput.ChainIndexingProposalStatusEntry

Field	Type	Description	Label
key	*int32*		
value	*PendingChainIndexingProposalStatus*		

CrossChain.GetPendingCrossChainIndexingProposalOutput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal id of cross chain indexing.	
proposer	*aelf.Address*	The proposer of cross chain indexing proposal.	
to_be_released	*bool*	True if the proposal can be released, otherwise false.	
proposed_cross_chain_block_data	*acs7.CrossChainBlockData*	The cross chain data proposed.	
expired_time	*google.protobuf.Timestamp*	The proposal expiration time.	

CrossChain.InitializeInput

Field	Type	Description	Label
parent_chain_id	*int32*	The id of parent chain.	
creation_height_on_parent_chain	*int64*	The height of side chain created on parent chain.	
is_privilege_preserved	*bool*	True if chain privilege needed, otherwise false.	

CrossChain.PendingChainIndexingProposalStatus

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of cross chain indexing proposal.	
proposer	*aelf.Address*	The proposer of cross chain indexing.	
to_be_released	*bool*	True if the proposal can be released, otherwise false.	
proposed_cross_chain_block_data	*acs7.CrossChainBlockData*	The cross chain data proposed.	
expired_time	*google.protobuf.Timestamp*	The proposal expiration time.	

CrossChain.ProposedCrossChainIndexing

Field	Type	Description	Label
chain_indexing_proposal_collections	*ProposedCrossChainIndexing.ChainIndexingProposalCollectionsEntry*	The collection of chain indexing proposal, the key is chain id.	repeated

CrossChain.ProposedCrossChainIndexing.ChainIndexingProposalCollectionsEntry

Field	Type	Description	Label
key	*int32*		
value	*ChainIndexingProposal*		

CrossChain.SideChainCreatedEvent

Field	Type	Description	Label
creator	*aelf.Address*	The proposer who propose to create the side chain.	
chainId	*int32*	The created side chain id.	

CrossChain.SideChainCreationRequestState

Field	Type	Description	Label
side_chain_creation_request	*acs7.SideChainCreationRequest*	The parameters of creating side chain.	
expired_time	*google.protobuf.Timestamp*	The expiration date of the proposal.	
proposer	*aelf.Address*	The proposer who proposed to create the side chain.	

CrossChain.SideChainIndexingFeeControllerChanged

Field	Type	Description	Label
chain_id	*int32*	The side chain id.	
authority_info	*AuthorityInfo*	The changed controller of side chain indexing fee.	

CrossChain.SideChainInfo

Field	Type	Description	Label
proposer	*aelf.Address*	The proposer who propose to create the side chain.	
side_chain_status	*SideChainStatus*	The status of side chain.	
side_chain_id	*int32*	The side chain id.	
creation_timestamp	*google.protobuf.Timestamp*	The time of side chain created.	
creation_height_on_parent_chain	*int64*	The height of side chain created on parent chain.	
indexing_price	*int64*	The price of indexing fee.	
is_privilege_preserved	*bool*	True if chain privilege needed, otherwise false.	
arrears_info	*SideChainInfo.ArrearsInfoEntry*	creditor and amounts for the chain indexing fee debt	repeated
indexing_fee_controller	*AuthorityInfo*	The controller of indexing fee.	

CrossChain.SideChainInfo.ArrearsInfoEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

CrossChain.SideChainLifetimeControllerChanged

Field	Type	Description	Label
authority_info	*AuthorityInfo*	The changed controller of side chain lifetime.	

CrossChain.CrossChainIndexingProposalStatus

Name	Number	Description
NON_PROPOSED	0	
PENDING	1	The proposal is pending.
ACCEPTED	2	The proposal has been released.

CrossChain.SideChainStatus

Name	Number	Description
FATAL	0	Currently no meaning.
ACTIVE	1	The side chain is being indexed.
INDEXING_FEE_DEBT	2	The side chain is in debt for indexing fee.
TERMINATED	3	The side chain is disposed.

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS7

acs7.AdjustIndexingFeeInput

Field	Type	Description	Label
side_chain_id	*int32*	The side chain id to adjust.	
indexing_fee	*int64*	The new price of indexing fee.	

acs7.ChainIdAndHeightDict

Field	Type	Description	Label
id_height_dict	*ChainIdAndHeightDict.IdHeightDictEntry*	A collection of chain ids and heights, where the key is the chain id and the value is the height.	repeated

acs7.ChainIdAndHeightDict.IdHeightDictEntry

Field	Type	Description	Label
key	*int32*		
value	*int64*		

acs7.ChainInitializationConsensusInfo

Field	Type	Description	Label
initial_consensus_data	*bytes*	Initial consensus data.	

acs7.ChainInitializationData

Field	Type	Description	Label
chain_id	*int32*	The id of side chain.	
creator	*aelf.Address*	The side chain creator.	
creation_timestamp	*google.protobuf.Timestamp*	The timestamp for side chain creation.	
creation_height_on_parent_chain	*int64*	The height of side chain creation on parent chain.	
chain_creator_privilege_preserved	*bool*	Creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.	
parent_chain_token_contract_address	*aelf.Address*	Parent chain token contract address.	
chain_initialization_consensus_info	*ChainInitializationConsensusInfo*	Initial consensus information.	
native_token_info_data	*bytes*	The native token info.	
resource_token_info	*ResourceTokenInfo*	The resource token information.	
chain_primary_token_info	*ChainPrimaryTokenInfo*	The chain primary token information.	

acs7.ChainPrimaryTokenInfo

Field	Type	Description	Label
chain_primary_token_data	*bytes*	The side chain primary token data.	
side_chain_token_initial_issue_list	*SideChainTokenInitialIssue*	The side chain primary token initial issue list.	repeated

acs7.CreateSideChainInput

Field	Type	Description	Label
side_chain_creation_request	*SideChainCreationRequest*	The request information of the side chain creation.	
proposer	*aelf.Address*	The proposer of the side chain creation.	

acs7.CrossChainBlockData

Field	Type	Description	Label
side_chain_block_data_list	*SideChainBlockData*	The side chain block data list to index.	repeated
parent_chain_block_data_list	*ParentChainBlockData*	The parent chain block data list to index.	repeated

acs7.CrossChainExtraData

Field	Type	Description	Label
transaction_status_merkle_tree_root	*aelf.Hash*	Merkle tree root of side chain block transaction status root.	

acs7.CrossChainIndexingDataProposedEvent

Field	Type	Description	Label
proposed_cross_chain_data	*CrossChainBlockData*	Proposed cross chain data to be indexed.	
proposal_id	*aelf.Hash*	The proposal id.	

acs7.CrossChainMerkleProofContext

Field	Type	Description	Label
bound_parent_chain_height	*int64*	The height of parent chain bound up with side chain.	
merkle_path_from_parent_chain	*aelf.MerklePath*	The merkle path generated from parent chain.	

acs7.IndexedParentChainBlockData

Field	Type	Description	Label
local_chain_height	*int64*	The height of the local chain when indexing the parent chain.	
parent_chain_block_data_list	*ParentChainBlockData*	Parent chain block data.	repeated

acs7.IndexedSideChainBlockData

Field	Type	Description	Label
side_chain_block_data_list	*SideChainBlockData*	Side chain block data.	repeated

acs7.ParentChainBlockData

Field	Type	Description	Label
height	*int64*	The height of parent chain.	
cross_chain_extra_data	*CrossChainExtraData*	The merkle tree root computing from side chain roots.	
chain_id	*int32*	The parent chain id.	
transaction_status_merkle_tree_root	*aelf.Hash*	The merkle tree root computing from transactions status in parent chain block.	
indexed_merkle_path	*ParentChainBlockData.IndexedMerklePathEntry*	Indexed block height from side chain and merkle path for this side chain block	repeated
extra_data	*ParentChainBlockData.ExtraDataEntry*	Extra data map.	repeated

acs7.ParentChainBlockData.ExtraDataEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

acs7.ParentChainBlockData.IndexedMerklePathEntry

Field	Type	Description	Label
key	*int64*		
value	*aelf.MerklePath*		

acs7.RechargeInput

Field	Type	Description	Label
chain_id	*int32*	The chain id to recharge.	
amount	*int64*	The amount to recharge.	

acs7.ReleaseCrossChainIndexingProposalInput

Field	Type	Description	Label
chain_id_list	*int32*	List of chain ids to release.	repeated

acs7.ReleaseSideChainCreationInput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal id of side chain creation.	

acs7.ResourceTokenInfo

Field	Type	Description	Label
resource_token_list_data	*bytes*	The resource token information.	
initial_resource_amount	*ResourceTokenInfo.InitialResourceAmountEntry*	The initial resource token amount.	repeated

acs7.ResourceTokenInfo.InitialResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

acs7.SideChainBlockData

Field	Type	Description	Label
height	*int64*	The height of side chain block.	
block_header_hash	*aelf.Hash*	The hash of side chain block.	
transaction_status_merkle_tree_root	*aelf.Hash*	The merkle tree root computing from transactions status in side chain block.	
chain_id	*int32*	The id of side chain.	

acs7.SideChainBlockDataIndexed

acs7.SideChainCreationRequest

Field	Type	Description	Label
indexing_price	*int64*	The cross chain indexing price.	
locked_token_amount	*int64*	Initial locked balance for a new side chain.	
is_privilege_preserved	*bool*	Creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.	
side_chain_token_creation_request	*SideChainTokenCreationRequest*	Side chain token information.	
side_chain_token_initial_issue_list	*SideChainTokenInitialIssue*	A list of accounts and amounts that will be issued when the chain starts.	repeated
initial_resource_amount	*SideChainCreationRequest.InitialResourceAmountEntry*	The initial rent resources.	repeated

acs7.SideChainCreationRequest.InitialResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

acs7.SideChainIndexingInformation

Field	Type	Description	Label
chain_id	*int32*	The side chain id.	
indexed_height	*int64*	The indexed height.	

acs7.SideChainIndexingInformationList

Field	Type	Description	Label
indexing_information_list	*SideChainIndexingInformation*	A list contains indexing information of side chains.	repeated

acs7.SideChainTokenCreationRequest

Field	Type	Description	Label
side_chain_token_symbol	*string*	Token symbol of the side chain to be created	
side_chain_token_name	*string*	Token name of the side chain to be created	
side_chain_token_total_supply	*int64*	Token total supply of the side chain to be created	
side_chain_token_decimals	*int32*	Token decimals of the side chain to be created	

acs7.SideChainTokenInitialIssue

Field	Type	Description	Label
address	*aelf.Address*	The account that will be issued.	
amount	*int64*	The amount that will be issued.	

acs7.VerifyTransactionInput

Field	Type	Description	Label
transaction_id	*aelf.Hash*	The cross chain transaction id to verify.	
path	*aelf.MerklePath*	The merkle path of the transaction.	
parent_chain_height	*int64*	The height of parent chain that indexing this transaction.	
verified_chain_id	*int32*	The chain if to verify.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.10 AElf.Contracts.Treasury

Treasury contract.

Used for distributing bonus' to voters and candidates during the election process.

Implement AElf Standards ACS1 and ACS10.

20.10.1 Contract Methods

Method Name	Request Type	Response Type	Description
InitialTreasuryContract	*google.protobuf.Empty*	*google.protobuf.Empty*	Initialize treasury contract.
InitialMiningRewardProfitItem	*google.protobuf.Empty*	*google.protobuf.Empty*	Initialize the sub-item of the bonus scheme.
DonateAll	*Treasury.DonateAllInput*	*google.protobuf.Empty*	Donate all tokens owned by the sender.
SetDividendPoolWeightSetting	*Treasury.DividendPoolWeightSetting*	*google.protobuf.Empty*	Set the dividend weight of the sub-item of the dividend item.
SetMinerRewardWeightSetting	*Treasury.MinerRewardWeightSetting*	*google.protobuf.Empty*	Set the miner reward weight.
UpdateMiningReward	*google.protobuf.Int64Value*	*google.protobuf.Empty*	Set the reward for mining.
ChangeTreasuryController	*AuthorityInfo*	*google.protobuf.Empty*	Change the governance authority information for treasury contract.
RecordMinerReplacement	*Treasury.RecordMinerReplacementInput*	*google.protobuf.Empty*	AEDPoS Contract can notify Treasury Contract to aware miner replacement happened.
GetWelfareRewardAmountSample	*Treasury.GetWelfareRewardAmountSampleInput*	*Treasury.GetWelfareRewardAmountSampleOutput*	Used to estimate the revenue weight of 10000 tokens voted by users.
GetTreasurySchemeId	*google.protobuf.Empty*	*aelf.Hash*	Get the scheme id of treasury.
GetDividendPoolWeightProportion	*google.protobuf.Empty*	*Treasury.DividendPoolWeightProportion*	Query the weight percentage of dividend pool items.
GetMinerRewardWeightProportion	*google.protobuf.Empty*	*Treasury.MinerRewardWeightProportion*	Query the weight percentage of the dividend item for miner.
GetTreasuryController	*google.protobuf.Empty*	*AuthorityInfo*	Query the governance authority information.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

AElf.Standards.ACS10

Method Name	Request Type	Response Type	Description
Donate	*acs10.DonateInput*	*google.protobuf.Empty*	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.
Release	*acs10.ReleaseInput*	*google.protobuf.Empty*	Release dividend pool according the period number.
SetSymbolList	*acs10.SymbolList*	*google.protobuf.Empty*	Set the token symbols dividend pool supports.
GetSymbolList	*google.protobuf.Empty*	*acs10.SymbolList*	Query the token symbols dividend pool supports.
GetUndistributedDividends	*google.protobuf.Empty*	*acs10.Dividends*	Query the balance of undistributed tokens whose symbols are included in the symbol list.
GetDividends	*google.protobuf.Int64Value*	*acs10.Dividends*	Query the dividend information according to the height.

20.10.2 Contract Types

AElf.Contracts.Treasury

Treasury.DividendPoolWeightProportion

Field	Type	Description	Label
citizen_welfare_proportion_info	*SchemeProportionInfo*	The proportion of citizen welfare.	
backup_subsidy_proportion_info	*SchemeProportionInfo*	The proportion of candidate nodes.	
miner_reward_proportion_info	*SchemeProportionInfo*	The proportion of miner	

Treasury.DividendPoolWeightSetting

Field	Type	Description	Label
citizen_welfare_weight	*int32*	The dividend weight of citizen welfare.	
backup_subsidy_weight	*int32*	The dividend weight of candidate nodes.	
miner_reward_weight	*int32*	The dividend weight of miner.	

Treasury.DonateAllInput

Field	Type	Description	Label
symbol	*string*	The token symbol to donate.	

Treasury.GetWelfareRewardAmountSampleInput

Field	Type	Description	Label
value	*int64*	Token lock time.	repeated

Treasury.GetWelfareRewardAmountSampleOutput

Field	Type	Description	Label
value	*int64*	The weight calculated.	repeated

Treasury.MinerReElectionInformation

Field	Type	Description	Label
continual_appointment_times	*MinerReElectionInformation.ContinualAppointmentTimesEntry*	The reappointment information for miner.	repeated

Treasury.MinerReElectionInformation.ContinualAppointmentTimesEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Treasury.MinerRewardWeightProportion

Field	Type	Description	Label
basic_miner_reward_proportion_info	*SchemeProportionInfo*	The proportion of the basic income of the miner.	
votes_weight_reward_proportion_info	*SchemeProportionInfo*	The proportion of the vote of the miner.	
re_election_reward_proportion_info	*SchemeProportionInfo*	The proportion of the reappointment of the miner.	

Treasury.MinerRewardWeightSetting

Field	Type	Description	Label
basic_miner_reward_weight	*int32*	The dividend weight of the basic income of the miner.	
votes_weight_reward_weight	*int32*	The dividend weight of the vote of the miner.	
re_election_reward_weight	*int32*	The dividend weight of the reappointment of the miner.	

Treasury.RecordMinerReplacementInput

Field	Type	Description	Label
old_pubkey	*string*		
new_pubkey	*string*		
current_term_number	*int64*		

Treasury.SchemeProportionInfo

Field	Type	Description	Label
scheme_id	*aelf.Hash*	The scheme id.	
proportion	*int32*	Dividend weight percentage.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Standards.ACS10

acs10.Dividends

Field	Type	Description	Label
value	*Dividends.ValueEntry*	The dividends, symbol -> amount.	repeated

acs10.Dividends.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs10.DonateInput

Field	Type	Description	Label
symbol	*string*	The token symbol to donate.	
amount	*int64*	The amount to donate.	

acs10.DonationReceived

Field	Type	Description	Label
from	*aelf.Address*	The address of donors.	
pool_contract	*aelf.Address*	The address of dividend pool.	
symbol	*string*	The token symbol Donated.	
amount	*int64*	The amount Donated.	

acs10.ReleaseInput

Field	Type	Description	Label
period_number	*int64*	The period number to release.	

acs10.SymbolList

Field	Type	Description	Label
value	*string*	The token symbol list.	repeated

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.11 AElf.Contracts.Vote

Vote contract.

The Vote contract is an abstract layer for voting. Developers implement concrete voting activities by calling this contract.

Implement AElf Standards ACS1.

20.11.1 Contract Methods

Method Name	Request Type	Response Type	Description
Register	*Vote.VotingRegisterInput*	*google.protobuf.Empty*	Create a voting activity.
Vote	*Vote.VoteInput*	*google.protobuf.Empty*	After successfully creating a voting activity, others are able to vote.
Withdraw	*Vote.WithdrawInput*	*google.protobuf.Empty*	A voter can withdraw the token after the lock time.
TakeSnapshot	*Vote.TakeSnapshotInput*	*google.protobuf.Empty*	Save the result of the specified number of votes and generates a new round votes.
AddOption	*Vote.AddOptionInput*	*google.protobuf.Empty*	Add an option to a voting activity.
RemoveOption	*Vote.RemoveOptionInput*	*google.protobuf.Empty*	Remove an option from a voting activity.
AddOptions	*Vote.AddOptionsInput*	*google.protobuf.Empty*	Add multiple options to a voting activity.
RemoveOptions	*Vote.RemoveOptionsInput*	*google.protobuf.Empty*	Remove multiple options from a voting activity.
GetVotingItem	*Vote.GetVotingItemInput*	*Vote.VotingItem*	Get a voting activity information.
GetVotingResult	*Vote.GetVotingResultInput*	*Vote.VotingResult*	Get a voting result according to the provided voting activity id and snapshot number.
GetLatestVotingResult	*aelf.Hash*	*Vote.VotingResult*	Gets the latest result according to the voting activity id.
GetVotingRecord	*aelf.Hash*	*Vote.VotingRecord*	Get the voting record according to vote id.
GetVotingRecords	*Vote.GetVotingRecordsInput*	*Vote.VotingRecords*	Get the voting record according to vote ids.
GetVotedItems	*aelf.Address*	*Vote.VotedItems*	Get all voted information according to voter address.
GetVotingIds	*Vote.GetVotingIdsInput*	*Vote.VotedIds*	Get the vote ids according to voting activity id.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.11.2 Contract Types

AElf.Contracts.Vote

Vote.AddOptionInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
option	*string*	The new option to add.	

Vote.AddOptionsInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
options	*string*	The new options to add.	repeated

Vote.GetVotingIdsInput

Field	Type	Description	Label
voter	*aelf.Address*	The address of voter.	
voting_item_id	*aelf.Hash*	The voting activity id.	

Vote.GetVotingItemInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	

Vote.GetVotingRecordsInput

Field	Type	Description	Label
ids	*aelf.Hash*	The vote ids.	repeated

Vote.GetVotingResultInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
snapshot_number	*int64*	The snapshot number.	

Vote.RemoveOptionInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
option	*string*	The option to remove.	

Vote.RemoveOptionsInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
options	*string*	The options to remove.	repeated

Vote.TakeSnapshotInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
snapshot_number	*int64*	The snapshot number to take.	

Vote.VoteInput

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
voter	*aelf.Address*	The address of voter.	
amount	*int64*	The amount of vote.	
option	*string*	The option to vote.	
vote_id	*aelf.Hash*	The vote id.	
is_change_target	*bool*	Whether vote others.	

Vote.Voted

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
voter	*aelf.Address*	The address of voter.	
snapshot_number	*int64*	The snapshot number.	
amount	*int64*	The amount of vote.	
vote_timestamp	*google.protobuf.Timestamp*	The time of vote.	
option	*string*	The option voted.	
vote_id	*aelf.Hash*	The vote id.	

Vote.VotedIds

Field	Type	Description	Label
active_votes	*aelf.Hash*	The active vote ids.	repeated
withdrawn_votes	*aelf.Hash*	The withdrawn vote ids.	repeated

Vote.VotedItems

Field	Type	Description	Label
voted_item_vote_ids	*VotedItems.VotedItemVoteIdsEntry*	The voted ids.	repeated

Vote.VotedItems.VotedItemVoteIdsEntry

Field	Type	Description	Label
key	*string*		
value	*VotedIds*		

Vote.VotingItem

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
accepted_currency	*string*	The token symbol which will be accepted.	
is_lock_token	*bool*	Whether the vote will lock token.	
current_snapshot_number	*int64*	The current snapshot number.	
total_snapshot_number	*int64*	The total snapshot number.	
options	*string*	The list of options.	repeated
register_timestamp	*google.protobuf.Timestamp*	The register time of the voting activity.	
start_timestamp	*google.protobuf.Timestamp*	The start time of the voting.	
end_timestamp	*google.protobuf.Timestamp*	The end time of the voting.	
current_snapshot_start_timestamp	*google.protobuf.Timestamp*	The start time of current round of the voting.	
sponsor	*aelf.Address*	The sponsor address of the voting activity.	

Vote.VotingItemRegistered

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
accepted_currency	*string*	The token symbol which will be accepted.	
is_lock_token	*bool*	Whether the vote will lock token.	
current_snapshot_number	*int64*	The current snapshot number.	
total_snapshot_number	*int64*	The total number of snapshots of the vote.	
register_timestamp	*google.protobuf.Timestamp*	The register time of the voting activity.	
start_timestamp	*google.protobuf.Timestamp*	The start time of the voting.	
end_timestamp	*google.protobuf.Timestamp*	The end time of the voting.	
current_snapshot_start_timestamp	*google.protobuf.Timestamp*	The start time of current round of the voting.	
sponsor	*aelf.Address*	The sponsor address of the voting activity.	

Vote.VotingRecord

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
voter	*aelf.Address*	The address of voter.	
snapshot_number	*int64*	The snapshot number.	
amount	*int64*	The amount of vote.	
withdraw_timestamp	*google.protobuf.Timestamp*	The time of withdraw.	
vote_timestamp	*google.protobuf.Timestamp*	The time of vote.	
is_withdrawn	*bool*	Whether the vote had been withdrawn.	
option	*string*	The option voted.	
is_change_target	*bool*	Whether vote others.	

Vote.VotingRecords

Field	Type	Description	Label
records	*VotingRecord*	The voting records.	repeated

Vote.VotingRegisterInput

Field	Type	Description	Label
start_timestamp	*google.protobuf.Timestamp*	The start time of the voting.	
end_timestamp	*google.protobuf.Timestamp*	The end time of the voting.	
accepted_currency	*string*	The token symbol which will be accepted.	
is_lock_token	*bool*	Whether the vote will lock token.	
total_snapshot_number	*int64*	The total number of snapshots of the vote.	
options	*string*	The list of options.	repeated

Vote.VotingResult

Field	Type	Description	Label
voting_item_id	*aelf.Hash*	The voting activity id.	
results	*VotingResult.ResultsEntry*	The voting result, option -> amount of votes,	repeated
snapshot_number	*int64*	The snapshot number.	
voters_count	*int64*	The total number of voters.	
snapshot_start_timestamp	*google.protobuf.Timestamp*	The start time of this snapshot.	
snapshot_end_timestamp	*google.protobuf.Timestamp*	The end time of this snapshot.	
votes_amount	*int64*	Total votes received during the process of this snapshot.	

Vote.VotingResult.ResultsEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

Vote.WithdrawInput

Field	Type	Description	Label
vote_id	*aelf.Hash*	The vote id.	

Vote.Withdrawn

Field	Type	Description	Label
vote_id	*aelf.Hash*	The vote id.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.12 AElf.Contracts.TokenHolder

TokenHolder contract.

Used to build a a bonus model for distributing bonus' to whom hold the token.

Implement AElf Standards ACS1.

20.12.1 Contract Methods

Method Name	Request Type	Response Type	Description
CreateScheme	*TokenHolder.CreateTokenHolderProfitSchemeInput*	*google.protobuf.Empty*	Create a scheme for distributing bonus.
AddBeneficiary	*TokenHolder.AddTokenHolderBeneficiaryInput*	*google.protobuf.Empty*	Add a beneficiary to a scheme.
RemoveBeneficiary	*TokenHolder.RemoveTokenHolderBeneficiaryInput*	*google.protobuf.Empty*	Removes a beneficiary from a scheme. Note: amount > 0: update the weight of the beneficiary, amount = 0: remove the beneficiary.
ContributeProfits	*TokenHolder.ContributeProfitsInput*	*google.protobuf.Empty*	Contribute profit to a scheme.
DistributeProfits	*TokenHolder.DistributeProfitsInput*	*google.protobuf.Empty*	To distribute the profits of the scheme, the stakeholders of the project may go to receive dividends.
RegisterForProfits	*TokenHolder.RegisterForProfitsInput*	*google.protobuf.Empty*	The user registers a bonus project.
Withdraw	*aelf.Address*	*google.protobuf.Empty*	After the lockup time expires, the user can withdraw token.
ClaimProfits	*TokenHolder.ClaimProfitsInput*	*google.protobuf.Empty*	After DistributeProfits the holder can get his dividend.
GetScheme	*aelf.Address*	*TokenHolder.TokenHolderProfitScheme*	Query the details of the specified scheme.
GetProfitsMap	*TokenHolder.ClaimProfitsInput*	*TokenHolder.ReceivedProfitsMap*	Query the dividends available to the holder.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.12.2 Contract Types

AElf.Contracts.TokenHolder

TokenHolder.AddTokenHolderBeneficiaryInput

Field	Type	Description	Label
beneficiary	*aelf.Address*	Beneficiary's address.	
shares	*int64*	The weight of the beneficiary's dividends in the scheme.	

TokenHolder.ClaimProfitsInput

Field	Type	Description	Label
scheme_manager	*aelf.Address*	The manager of the scheme.	
beneficiary	*aelf.Address*	Beneficiary's address.	

TokenHolder.ContributeProfitsInput

Field	Type	Description	Label
scheme_manager	*aelf.Address*	The manager of the scheme.	
amount	*int64*	The amount of token to contribute.	
symbol	*string*	The symbol of token to contribute.	

TokenHolder.CreateTokenHolderProfitSchemeInput

Field	Type	Description	Label
symbol	*string*	The token symbol.	
minimum_lock_minutes	*int64*	Minimum lock time for holding token.	
auto_distribute_threshold	*CreateTokenHolderProfitSchemeInput.AutoDistributeThresholdEntry*	Threshold setting for releasing dividends.	repeated

TokenHolder.CreateTokenHolderProfitSchemeInput.AutoDistributeThresholdEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

TokenHolder.DistributeProfitsInput

Field	Type	Description	Label
scheme_manager	*aelf.Address*	The manager of the scheme.	
amounts_map	*DistributeProfitsInput.AmountsMapEntry*	The token to distribute, symbol -> amount.	repeated

TokenHolder.DistributeProfitsInput.AmountsMapEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

TokenHolder.ReceivedProfitsMap

Field	Type	Description	Label
value	*ReceivedProfitsMap.ValueEntry*	The amount of token the beneficiary can get, symbol -> amount.	repeated

TokenHolder.ReceivedProfitsMap.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

TokenHolder.RegisterForProfitsInput

Field	Type	Description	Label
scheme_manager	*aelf.Address*	The manager of the scheme.	
amount	*int64*	The amount of token holding.	

TokenHolder.RemoveTokenHolderBeneficiaryInput

Field	Type	Description	Label
beneficiary	*aelf.Address*	Beneficiary's address.	
amount	*int64*	The amount of weights to remove.	

TokenHolder.TokenHolderProfitScheme

Field	Type	Description	Label
symbol	*string*	The token symbol.	
scheme_id	*aelf.Hash*	The scheme id.	
period	*int64*	The current dividend period.	
minimum_lock_minutes	*int64*	Minimum lock time for holding token.	
auto_distribute_threshold	*TokenHolderProfitScheme.AutoDistributeThresholdEntry*	Threshold setting for releasing dividends.	repeated

TokenHolder.TokenHolderProfitScheme.AutoDistributeThresholdEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.13 AElf.Contracts.Economic

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.

Implement AElf Standards ACS1.

20.13.1 Contract Methods

Method Name	Request Type	Response Type	Description
IssueNativeToken	*Economic.IssueNativeTokenInput*	*google.protobuf.Empty*	Only ZeroContract is able to issue the native token.
InitialEconomicSystem	*Economic.InitialEconomicSystemInput*	*google.protobuf.Empty*	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.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.13.2 Contract Types

AElf.Contracts.Economic

Economic.InitialEconomicSystemInput

Field	Type	Description	Label
native_token_symbol	*string*	The native token symbol.	
native_token_name	*string*	The native token name.	
native_token_total_supply	*int64*	The native token total supply.	
native_token_decimals	*int32*	The accuracy of the native token.	
is_native_token_burnable	*bool*	It indicates if the token is burnable.	
mining_reward_total_amount	*int64*	It determines how much native token is used to reward the miners.	
transaction_size_fee_unit_price	*int64*	todo : remove unused fields	

Economic.IssueNativeTokenInput

Field	Type	Description	Label
amount	*int64*	The amount of token.	
memo	*string*	The memo.	
to	*aelf.Address*	The recipient of the token.	

Economic.IssueResourceTokenInput

Field	Type	Description	Label
symbol	*string*	The symbol of resource token.	
amount	*int64*	The amount of resource token.	
memo	*string*	The memo.	
to	*aelf.Address*	The recipient of the token.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.14 AElf.Contracts.TokenConverter

TokenConvert contract.

Using this contract can build a connection between the base 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.

Implement AElf Standards ACS1.

20.14.1 Contract Methods

Method Name	Request Type	Response Type	Description
Initialize	*TokenConverter.InitializeInput*	*google.protobuf.Empty*	Initialize TokenConvert contract.
SetConnector	*TokenConverter.Connector*	*google.protobuf.Empty*	
Buy	*TokenConverter.BuyInput*	*google.protobuf.Empty*	After establishing bancor model of token and base token, you can buy token through this method.
Sell	*TokenConverter.SellInput*	*google.protobuf.Empty*	After establishing bancor model of token and base token, you can sell token through this method.
SetFeeRate	*google.protobuf.StringValue*	*google.protobuf.Empty*	Set the fee rate for buy/sell (fee amount = cost * feeRate).
UpdateConnector	*TokenConverter.Connector*	*google.protobuf.Empty*	Before calling the EnableConnector, the connector controller can update the pair connector through this method.
AddPairConnector	*TokenConverter.PairConnectorParam*	*google.protobuf.Empty*	Add a pair connector for new token and the base token.
EnableConnector	*TokenConverter.ToBeConnectedTokenInfo*	*google.protobuf.Empty*	After adding a pair, you need to call this method to enable it before buy and sell token.
ChangeConnectorController	*AuthorityInfo*	*google.protobuf.Empty*	Set the governance authority information for TokenConvert contract.
GetPairConnector	*TokenConverter.TokenSymbol*	*TokenConverter.PairConnector*	Query the pair connector according to token symbol.
GetFeeRate	*google.protobuf.Empty*	*google.protobuf.StringValue*	Query the fee rate for buy/sell.
GetBaseTokenSymbol	*google.protobuf.Empty*	*TokenConverter.TokenSymbol*	Query the symbol of base token.
GetNeededDeposit	*TokenConverter.ToBeConnectedTokenInfo*	*TokenConverter.DepositInfo*	Query how much the base token need be deposited before enabling the connector.
GetDepositConnectorBalance	*google.protobuf.StringValue*	*google.protobuf.Int64Value*	Query how much the base token have been deposited.
GetControllerForManageConnector	*google.protobuf.Empty*	*AuthorityInfo*	Query the governance authority information for TokenConvert contract.
IsSymbolAbleToSell	*google.protobuf.StringValue*	*google.protobuf.BoolValue*	Query whether the token can be sold.

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.14.2 Contract Types

AElf.Contracts.TokenConverter

TokenConverter.BuyInput

Field	Type	Description	Label
symbol	*string*	The token symbol you want to buy.	
amount	*int64*	The amount you want to buy.	
pay_limit	*int64*	Limit of cost. If the token required for buy exceeds this value, the buy will be abandoned. And 0 is no limit.	

TokenConverter.Connector

Field	Type	Description	Label
symbol	*string*	The token symbol.	
virtual_balance	*int64*	The virtual balance for base token.	
weight	*string*	The calculated weight value for this Connector.	
is_virtual_balance_enabled	*bool*	Whether to use Virtual Balance.	
is_purchase_enabled	*bool*	Whether the connector is enabled.	
related_symbol	*string*	Indicates its related connector, the pair connector includes a new created token connector and the base token connector.	
is_deposit_account	*bool*	Indicates if the connector is base token connector.	

TokenConverter.DepositInfo

Field	Type	Description	Label
need_amount	*int64*	How much more base Token is needed as the deposit.	
amount_out_of_token_convert	*int64*	How many tokens are not on the TokenConvert address.	

TokenConverter.InitializeInput

Field	Type	Description	Label
base_token_symbol	*string*	Base token symbol, default is the native token symbol.	
fee_rate	*string*	The fee rate for buy/sell.	
connectors	*Connector*	The default added connectors.	repeated

TokenConverter.PairConnector

Field	Type	Description	Label
resource_connector	*Connector*	The connector of the specified token.	
deposit_connector	*Connector*	The related connector.	

TokenConverter.PairConnectorParam

Field	Type	Description	Label
resource_connector_symbol	*string*	The token symbol.	
resource_weight	*string*	The weight value of this token in the Bancor model.	
native_virtual_balance	*int64*	This token corresponds to the value of base token.	
native_weight	*string*	The weight value of base token in Bancor model.	

TokenConverter.SellInput

Field	Type	Description	Label
symbol	*string*	The token symbol you want to sell.	
amount	*int64*	The amount you want to sell.	
receive_limit	*int64*	Limits on tokens obtained by selling. If the token obtained is less than this value, the sale will be abandoned. And 0 is no limit.	

TokenConverter.ToBeConnectedTokenInfo

Field	Type	Description	Label
token_symbol	*string*	The token symbol.	
amount_to_token_convert	*int64*	Specifies the number of tokens to convert to the TokenConvert contract.	

TokenConverter.TokenBought

Field	Type	Description	Label
symbol	*string*	The token symbol bought.	
bought_amount	*int64*	The amount bought.	
base_amount	*int64*	The total cost of the base token.	
fee_amount	*int64*	The fee amount.	

TokenConverter.TokenSold

Field	Type	Description	Label
symbol	*string*	The token symbol sold.	
sold_amount	*int64*	The amount sold.	
base_amount	*int64*	The total received of the base token.	
fee_amount	*int64*	The fee amount.	

TokenConverter.TokenSymbol

Field	Type	Description	Label
symbol	*string*	The token symbol.	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

20.15 AElf.Contracts.Configuration

Configuration contract.

Used to manage the configuration on the block chain.

Implement AElf Standards ACS1.

20.15.1 Contract Methods

Method Name	Request Type	Response Type	Description
SetConfiguration	*Configuration.SetConfigurationInput*	*google.protobuf.Empty*	Add or update configuration.
ChangeConfigurationController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is Parliament.
GetConfiguration	*google.protobuf.StringValue*	*google.protobuf.BytesValue*	Query the configuration by configuration's key.
GetConfigurationController	*google.protobuf.Empty*	*AuthorityInfo*	Query the controller information

AElf.Standards.ACS1

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

20.15.2 Contract Types

AElf.Contracts.Configuration

Configuration.ConfigurationSet

Field	Type	Description	Label
key	*string*	The configuration's key.	
value	*bytes*	The configuration's value(binary data).	

Configuration.SetConfigurationInput

Field	Type	Description	Label
key	*string*	The configuration's key.	
value	*bytes*	The configuration's value(binary data).	

AElf.Standards.ACS1

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AElf.Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

CHAPTER 21

Acs Introduction

21.1 ACS0 - Contract Deployment Standard

ACS0 is used to manage the deployment and update of contracts.

21.1.1 Interface

The contract inherited from ACS0 need implement the following interfaces:

Methods

Method Name	Request Type	Response Type	Description
DeploySystemSmartContract	*acs0.SystemContractDeploymentInput*	*aelf.Address*	Deploy a system smart contract on chain and return the address of the system contract deployed.
DeploySmartContract	*acs0.ContractDeploymentInput*	*aelf.Address*	Deploy a smart contract on chain and return the address of the contract deployed.
UpdateSmartContract	*acs0.ContractUpdateInput*	*aelf.Address*	Update a smart contract on chain.
ProposeNewContract	*acs0.ContractDeploymentInput*	*aelf.Hash*	Create a proposal to deploy a new contract and returns the id of the proposed contract.
ProposeContractCodeCheck	*acs0.ContractCodeCheckInput*	*aelf.Hash*	Create a proposal to check the code of a contract and return the id of the proposed contract.
ProposeUpdateContract	*acs0.ContractUpdateInput*	*aelf.Hash*	Create a proposal to update the specified contract and return the id of the proposed contract.
ReleaseApprovedContract	*acs0.ReleaseContractInput*	*google.protobuf.Empty*	Release the contract proposal which has been approved.
ReleaseCodeCheckedContract	*acs0.ReleaseContractInput*	*google.protobuf.Empty*	Release the proposal which has passed the code check.
ValidateSystemContractAddress	*acs0.ValidateSystemContractAddressInput*	*google.protobuf.Empty*	Validate whether the input system contract exists.
SetContractProposerRequiredState	*google.protobuf.BoolValue*	*google.protobuf.Empty*	Set authority of contract deployment.
CurrentContractSerialNumber	*google.protobuf.Empty*	*google.protobuf.Int64Value*	Get the current serial number of genesis contract (corresponds to the serial number that will be given to the next deployed contract).
GetContractInfo	*aelf.Address*	*acs0.ContractInfo*	Get detailed information about the specified contract.
GetContractAuthor	*aelf.Address*	*aelf.Address*	Get author of the specified contract.
GetContractHash	*aelf.Address*	*aelf.Hash*	Get the code hash of the contract about the specified address.
GetContractAddressByName	*aelf.Hash*	*aelf.Address*	Get the address of a system contract by its name.
GetSmartContractRegistrationByAddress	*aelf.Address*	*aelf.SmartContractRegistration*	Get the registration of a smart contract by its address.
GetSmartContractRegistrationByCodeHash	*aelf.Hash*	*aelf.SmartContractRegistration*	Get the registration of a smart contract by code hash.

Types

acs0.CodeCheckRequired

Field	Type	Description	Label
code	*bytes*	The byte array of the contract code.	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	
category	*sint32*	The category of contract code(0: C#).	
is_system_contract	*bool*	Indicates if the contract is the system contract.	

acs0.CodeUpdated

Field	Type	Description	Label
address	*aelf.Address*	The address of the updated contract.	
old_code_hash	*aelf.Hash*	The byte array of the old contract code.	
new_code_hash	*aelf.Hash*	The byte array of the new contract code.	
version	*int32*	The version of the current contract.	

acs0.ContractCodeCheckInput

Field	Type	Description	Label
contract_input	*bytes*	The byte array of the contract code to be checked.	
is_contract_deployment	*bool*	Whether the input contract is to be deployed or updated.	
code_check_release_method	*string*	Method to call after code check complete(DeploySmartContract or UpdateSmartContract).	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	
category	*sint32*	The category of contract code(0: C#).	
is_system_contract	*bool*	Indicates if the contract is the system contract.	

acs0.ContractDeployed

Field	Type	Description	Label
author	*aelf.Address*	The author of the contract, this is the person who deployed the contract.	
code_hash	*aelf.Hash*	The hash of the contract code.	
address	*aelf.Address*	The address of the contract.	
version	*int32*	The version of the current contract.	
Name	*aelf.Hash*	The name of the contract. It has to be unique.	

acs0.ContractDeploymentInput

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	

acs0.ContractInfo

Field	Type	Description	Label
serial_number	*int64*	The serial number of the contract.	
author	*aelf.Address*	The author of the contract, this is the person who deployed the contract.	
category	*sint32*	The category of contract code(0: C#).	
code_hash	*aelf.Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

acs0.ContractProposed

Field	Type	Description	Label
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	

acs0.ContractUpdateInput

Field	Type	Description	Label
address	*aelf.Address*	The contract address that needs to be updated.	
code	*bytes*	The byte array of the new contract code.	

acs0.ReleaseContractInput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The hash of the proposal.	
proposed_contract_input_hash	*aelf.Hash*	The id of the proposed contract.	

acs0.SystemContractDeploymentInput

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
name	*aelf.Hash*	The name of the contract. It has to be unique.	
transaction_method_call_list	*SystemContractDeploymentInput.SystemTransactionMethodCallList*	An initial list of transactions for the system contract, which is executed in sequence when the contract is deployed.	

acs0.SystemContractDeploymentInput.SystemTransactionMethodCall

Field	Type	Description	Label
method_name	*string*	The method name of system transaction.	
params	*bytes*	The params of system transaction method.	

acs0.SystemContractDeploymentInput.SystemTransactionMethodCallList

Field	Type	Description	Label
value	*SystemContractDeploymentInput.SystemTransactionMethodCall*	The list of system transactions.	repeated

acs0.ValidateSystemContractAddressInput

Field	Type	Description	Label
system_contract_hash_name	*aelf.Hash*	The name hash of the contract.	
address	*aelf.Address*	The address of the contract.	

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.1.2 Example

ACS0 declares methods for the scenes about contract deployment and update. AElf provides the implementation for ACS0, `Genesis Contract`. You can refer to the implementation of the *Genesis contract api*.

21.2 ACS1 - Transaction Fee Standard

ACS1 is used to manage the transfer fee.

21.2.1 Interface

The contract inherited from ACS1 need implement the APIs below:

Methods

Method Name	Request Type	Response Type	Description
SetMethodFee	*acs1.MethodFees*	*google.protobuf.Empty*	Set the method fees for the specified method. Note that this will override all fees of the method.
ChangeMethodFeeController	*AuthorityInfo*	*google.protobuf.Empty*	Change the method fee controller, the default is parliament and default organization.
GetMethodFee	*google.protobuf.StringValue*	*acs1.MethodFees*	Query method fee information by method name.
GetMethodFeeController	*google.protobuf.Empty*	*AuthorityInfo*	Query the method fee controller.

Types

acs1.MethodFee

Field	Type	Description	Label
symbol	*string*	The token symbol of the method fee.	
basic_fee	*int64*	The amount of fees to be charged.	

acs1.MethodFees

Field	Type	Description	Label
method_name	*string*	The name of the method to be charged.	
fees	*MethodFee*	List of fees to be charged.	repeated
is_size_fee_free	*bool*	Optional based on the implementation of SetMethodFee method.	

AuthorityInfo

Field	Type	Description	Label
contract_address	*aelf.Address*	The contract address of the controller.	
owner_address	*aelf.Address*	The address of the owner of the contract.	

Attention: just the system contract on main chain is able to implement acs1.

21.2.2 Usage

On AElf, a pre-transaction is generated by pre-plugin `FeeChargePreExecutionPlugin` before the transaction main processing. 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>
/// <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.");
```

(continues on next page)

(continued from previous page)

```
        }
    }
    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 parameter'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
```

(continues on next page)

(continued from previous page)

```
    {
        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
            {
                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.

21.2.3 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,
```

(continues on next page)

(continued from previous page)

```
            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` with type `AuthorityInfo`(in authority_info.proto)

```
public SingletonState<AuthorityInfo> MethodFeeController { get; set; }
```

Then, check the sender's right by comparing its address with owner.

```
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 Parliament organization. If a method is sent through this organization, 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);
   }
   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 `Parliament 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 with a fixed value, the other part of method is not to set method fee.

21.2.4 Test

Create ACS1's Stub, and call `GetMethodFee` and `GetMethodFeeController` to check if the return value is expected.

21.2.5 Example

All AElf system contracts implement ACS1, which can be used as a reference.

21.3 ACS2 - Parallel Execution Standard

ACS2 is used to provide information for parallel execution of transactions.

21.3.1 Interface

A contract that inherits ACS2 only needs to implement one method:

Methods

Method Name	Request Type	Response Type	Description
GetResource-Info	*aelf.Transaction*	*acs2.ResourceInfo*	Gets the resource information that the transaction execution depends on.

Types

acs2.ResourceInfo

Field	Type	Description	Label
write_paths	*aelf.ScopedStatePath*	The state path that depends on when writing.	repeated
read_paths	*aelf.ScopedStatePath*	The state path that depends on when reading.	repeated
non_parallelizable	*bool*	Whether the transaction is not executed in parallel.	

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.3.2 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` defines a property,

```
public MappedState<Address, string, long> Balances { get; set; }
```

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 .

21.3.3 Implementation

Token contract, as an example, 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
        {
```

(continues on next page)

(continued from previous page)

```
            Parts =
            {
                parts
            }
        }
    }
}
```

21.3.4 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 whether 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.
{
    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.
```

(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[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
    });
    groupedTransactions.Parallelizables.Count.ShouldBe(1);
}
```

21.3.5 Example

You can refer to the implementation of the `MultiToken contract` for `GetResourceInfo`. Noting that for the `ResourceInfo` provided by the method `Transfer`, you need to consider charging a transaction fee in addition to the two keys mentioned in this article.

21.4 ACS3 - Contract Proposal Standard

ACS3 is suitable for the case that a method needs to be approved by multiple parties. At this time, you can consider using some of the interfaces provided by ACS3.

21.4.1 Interface

If you want multiple addresses vote to get agreement to do something, you can implement the following methods defined in ACS3:

Methods

Method Name	Request Type	Response Type	Description
CreateProposal	*acs3.CreateProposalInput*	*aelf.Hash*	Create a proposal for which organization members can vote. When the proposal is released, a transaction will be sent to the specified contract. Return id of the newly created proposal.
Approve	*aelf.Hash*	*google.protobuf.Empty*	Approve a proposal according to the proposal ID.
Reject	*aelf.Hash*	*google.protobuf.Empty*	Reject a proposal according to the proposal ID.
Abstain	*aelf.Hash*	*google.protobuf.Empty*	Abstain a proposal according to the proposal ID.
Release	*aelf.Hash*	*google.protobuf.Empty*	Release a proposal according to the proposal ID and send a transaction to the specified contract.
ChangeOrganizationThreshold	*acs3.ProposalReleaseThreshold*	*google.protobuf.Empty*	Change 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.
ChangeOrganizationProposerWhiteList	*acs3.ProposerWhiteList*	*google.protobuf.Empty*	Change the white list of organization proposer. This method overrides the list of whitelisted proposers.
CreateProposalBySystemContract	*acs3.CreateProposalBySystemContractInput*	*aelf.Hash*	Create a proposal by system contracts, and return id of the newly created proposal.
ClearProposal	*aelf.Hash*	*google.protobuf.Empty*	Remove the specified proposal. If the proposal is in effect, the cleanup fails.
GetProposal	*aelf.Hash*	*acs3.ProposalOutput*	Get the proposal according to the proposal ID.
ValidateOrganizationExist	*aelf.Address*	*google.protobuf.BoolValue*	Check the existence of an organization.
ValidateProposerInWhiteList	*acs3.ValidateProposerInWhiteListInput*	*google.protobuf.BoolValue*	Check if the proposer is whitelisted.

Types

acs3.CreateProposalBySystemContractInput

Field	Type	Description	Label
proposal_input	*CreateProposalInput*	The parameters of creating proposal.	
origin_proposer	*aelf.Address*	The actor that trigger the call.	

acs3.CreateProposalInput

Field	Type	Description	Label
contract_method_name	*string*	The name of the method to call after release.	
to_address	*aelf.Address*	The address of the contract to call after release.	
params	*bytes*	The parameter of the method to be called after the release.	
expired_time	*google.protobuf.Timestamp*	The timestamp at which this proposal will expire.	
organization_address	*aelf.Address*	The address of the organization.	
proposal_description_url	*string*	Url is used for proposal describing.	
token	*aelf.Hash*	The token is for proposal id generation and with this token, proposal id can be calculated before proposing.	

acs3.OrganizationCreated

Field	Type	Description	Label
organization_address	*aelf.Address*	The address of the created organization.	

acs3.OrganizationHashAddressPair

Field	Type	Description	Label
organization_hash	*aelf.Hash*	The id of organization.	
organization_address	*aelf.Address*	The address of organization.	

acs3.OrganizationThresholdChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address	
proposer_release_threshold	*ProposalReleaseThreshold*	The new release threshold.	

acs3.OrganizationWhiteListChanged

Field	Type	Description	Label
organization_address	*aelf.Address*	The organization address.	
proposer_white_list	*ProposerWhiteList*	The new proposer whitelist.	

acs3.ProposalCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the created proposal.	
organization_address	*aelf.Address*	The organization address of the created proposal.	

acs3.ProposalOutput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
contract_method_name	*string*	The method that this proposal will call when being released.	
to_address	*aelf.Address*	The address of the target contract.	
params	*bytes*	The parameters of the release transaction.	
expired_time	*google.protobuf.Timestamp*	The date at which this proposal will expire.	
organization_address	*aelf.Address*	The address of this proposals organization.	
proposer	*aelf.Address*	The address of the proposer of this proposal.	
to_be_released	*bool*	Indicates if this proposal is releasable.	
approval_count	*int64*	Approval count for this proposal.	
rejection_count	*int64*	Rejection count for this proposal.	
abstention_count	*int64*	Abstention count for this proposal.	

acs3.ProposalReleaseThreshold

Field	Type	Description	Label
minimal_approval_threshold	*int64*	The value for the minimum approval threshold.	
maximal_rejection_threshold	*int64*	The value for the maximal rejection threshold.	
maximal_abstention_threshold	*int64*	The value for the maximal abstention threshold.	
minimal_vote_threshold	*int64*	The value for the minimal vote threshold.	

acs3.ProposalReleased

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the released proposal.	
organization_address	*aelf.Address*	The organization address of the released proposal.	

acs3.ProposerWhiteList

Field	Type	Description	Label
proposers	*aelf.Address*	The address of the proposers	repeated

acs3.ReceiptCreated

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The id of the proposal.	
address	*aelf.Address*	The sender address.	
receipt_type	*string*	The type of receipt(Approve, Reject or Abstain).	
time	*google.protobuf.Timestamp*	The timestamp of this method call.	
organization_address	*aelf.Address*	The address of the organization.	

acs3.ValidateProposerInWhiteListInput

Field	Type	Description	Label
proposer	*aelf.Address*	The address to search/check.	
organization_address	*aelf.Address*	The address of the organization.	

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.4.2 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
    };
    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);
```

(continues on next page)

(continued from previous page)

```
    State.Proposals[input] = proposal;
    return new Empty();
}
public override Empty Reject(Hash input)
{
    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;
```

(continues on next page)

(continued from previous page)

```
    return abstentionMemberCount > State.ProposalReleaseThreshold.Value.
↪MaximalAbstentionThreshold;
}
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;
}
```

21.4.3 Test

Before testing, two methods were added to a Dapp contract. We will test the proposal with these methods.

Define a singleton string and an organization address state in the State class:

```
public StringState Slogan { get; set; }
public SingletonState<Address> Organization { get; set; }
```

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 == State.Organization.Value, "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
{
    OrganizationAddress = OrganizationAddress
    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");
}
```

21.5 ACS4 - Consensus Standard

ACS4 is used to customize consensus mechanisms.

21.5.1 Interface

If you want to customize the consensus mechanism, you need to implement the following five interfaces:

Methods

Method Name	Request Type	Response Type	Description
GetConsensusCommand	*google.protobuf.BytesValue*	*acs4.ConsensusCommand*	Generate a consensus command based on the consensus contract state and the input public key.
GetConsensusExtraData	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Generate consensus extra data when a block is generated.
GenerateConsensusTransactions	*google.protobuf.BytesValue*	*acs4.TransactionList*	Generate consensus system transactions 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	*google.protobuf.BytesValue*	*acs4.ValidationResult*	Before executing the block, verify that the consensus information in the block header is correct.
ValidateConsensusAfterExecution	*google.protobuf.BytesValue*	*acs4.ValidationResult*	After executing the block, verify that the state information written to the consensus is correct.

Types

acs4.ConsensusCommand

Field	Type	Description	Label
limit_milliseconds_of_mining_block	*int32*	Time limit of mining next block.	
hint	*bytes*	Context of Hint is diverse according to the consensus protocol we choose, so we use bytes.	
arranged_mining_time	*google.protobuf.Timestamp*	The time of arrange mining.	
mining_due_time	*google.protobuf.Timestamp*	The expiration time of mining.	

acs4.TransactionList

Field	Type	Description	Label
transactions	*aelf.Transaction*	Consensus system transactions.	repeated

acs4.ValidationResult

Field	Type	Description	Label
success	*bool*	Is successful.	
message	*string*	The error message.	
is_re_trigger	*bool*	Whether to trigger mining again.	

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.5.2 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[]> GetConsensus ExtraDataAsync(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 GetConsensusExtraData.	At the time that the node produces a new block.

21.5.3 Example

You can refer to the implementation of the *AEDPoS contract*.

21.6 ACS5 - Contract Threshold Standard

If you want to raise the threshold for using contract, consider implementing ACS5.

21.6.1 Interface

To limit to call a method in a contract, you only need to implement the following five interfaces:

Methods

Method Name	Request Type	Response Type	Description
SetMethodCallingThreshold	*acs5.SetMethodCallingThresholdInput*	*google.protobuf.Empty*	Set the threshold for method calling.
GetMethodCallingThreshold	*google.protobuf.StringValue*	*acs5.MethodCallingThreshold*	Get the threshold for method calling.

Types

acs5.MethodCallingThreshold

Field	Type	Description	Label
symbol_to_amount	*MethodCallingThreshold.SymbolToAmountEntry*	The threshold for method calling, token symbol -> amount.	repeated
threshold_check_type	*ThresholdCheckType*	The type of threshold check.	

acs5.MethodCallingThreshold.SymbolToAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs5.SetMethodCallingThresholdInput

Field	Type	Description	Label
method	*string*	The method name to check.	
symbol_to_amount	*SetMethodCallingThresholdInput.SymbolToAmountEntry*	The threshold for method calling, token symbol -> amount.	repeated
threshold_check_type	*ThresholdCheckType*	The type of threshold check.	

acs5.SetMethodCallingThresholdInput.SymbolToAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs5.ThresholdCheckType

Name	Number	Description
BALANCE	0	Check balance only.
ALLOWANCE	1	Check balance and allowance at the same time.

21.6.2 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;
```

(continues on next page)

(continued from previous page)

```
                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.

21.6.3 Implementation

Just lik the `GetMethodFee` of ACS1, you can implement only one `GetMethodCallingThreshold` method.

It can also be achieved by using MappedState<string, MethodCallingThreshold> in the State class:

```
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();
}
```

(continues on next page)

(continued from previous page)

```
message SetMethodCallingThresholdInput {
    string method = 1;
    map<string, int64> symbol_to_amount = 2;// The order matters.
    ThresholdCheckType threshold_check_type = 3;
}
```

21.6.4 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());
```

(continues on next page)

(continued from previous page)

```
    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.");
}
```

21.7 ACS6 - Random Number Provider Standard

If your contract is about to generate a random number, you can consider using acs6.

21.7.1 Interface

To provider a random number according to certain input, you only need to implement one interface:

Methods

Method Name	Request Type	Response Type	Description
GetRandom-Bytes	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Get the random number provided by this contract.

21.7.2 Usage

All you need is to override this method to return a random number according to the given input. You can decide the certain logic of generating random number yourself, just remember to return a BytesValue type, thus the caller can deserialize the output himself.

21.7.3 Implementation

The easiest implementation

```
public override BytesValue GetRandomBytes(BytesValue input)
{
    var serializedInput = new GetRandomBytesInput();
    serializedInput.MergeFrom(input.Value);
    var value = new Hash();
    value.MergeFrom(serializedInput.Value);
    var randomHashFromContext = Context.GetRandomHash(value);
```

(continues on next page)

(continued from previous page)

```
    return new BytesValue
    {
        Value = serializedInput.Kind == 1
            ? new BytesValue {Value = randomHashFromContext.Value}.ToByteString()
            : new Int64Value {Value = Context.
↪ConvertHashToInt64(randomHashFromContext, 1, 10000)}.ToByteString()
    };
}
```

21.8 ACS7 - Contract CrossChain Standard

ACS7 is for cross chain related contract implementation.

21.8.1 Interface

This involves methods for chain creation and indexing:

Methods

Method Name	Request Type	Response Type	Description
ProposeCrossChainIndexing	*acs7.CrossChainBlockData*	*google.protobuf.Empty*	Propose once cross chain indexing.
ReleaseCrossChainIndexingProposal	*acs7.ReleaseCrossChainIndexingProposalInput*	*google.protobuf.Empty*	Release the proposed indexing if already approved.
RequestSideChainCreation	*acs7.SideChainCreationRequest*	*google.protobuf.Empty*	Request side chain creation.
ReleaseSideChainCreation	*acs7.ReleaseSideChainCreationInput*	*google.protobuf.Empty*	Release the side chain creation request if already approved and it will call the method CreateSideChain.
CreateSideChain	*acs7.CreateSideChainInput*	*google.protobuf.Int32Value*	Create the side chain and returns the newly created side chain ID. Only SideChainLifetimeController is permitted to invoke this method.
Recharge	*acs7.RechargeInput*	*google.protobuf.Empty*	Recharge for the specified side chain.
DisposeSideChain	*google.protobuf.Int32Value*	*google.protobuf.Int32Value*	Dispose a side chain according to side chain id. Only SideChainLifetimeController is permitted to invoke this method.
AdjustIndexingFeePrice	*acs7.AdjustIndexingFeeInput*	*google.protobuf.Empty*	Adjust side chain indexing fee. Only IndexingFeeController is permitted to invoke this method.
VerifyTransaction	*acs7.VerifyTransactionInput*	*google.protobuf.BoolValue*	Verify cross chain transaction.
GetSideChainIdAndHeight	*google.protobuf.Empty*	*acs7.ChainIdAndHeightDict*	Gets all the side chain id and height of the current chain.
GetSideChainIndexingInformationList	*google.protobuf.Empty*	*acs7.SideChainIndexingInformationList*	Get indexing information of side chains.
GetAllChainsIdAndHeight	*google.protobuf.Empty*	*acs7.ChainIdAndHeightDict*	Get id and recorded height of all chains.
GetIndexedSideChainBlockDataByHeight	*google.protobuf.Int64Value*	*acs7.IndexedSideChainBlockData*	Get block data of indexed side chain according to height.
GetBoundParentChainHeightAndMerklePathByHeight	*google.protobuf.Int64Value*	*acs7.CrossChainMerkleProofContext*	Get the merkle path bound up with side chain according to height.
GetChainInitializationData	*google.protobuf.Int32Value*	*acs7.ChainInitializationData*	Get initialization data for specified side chain.

Types

acs7.AdjustIndexingFeeInput

Field	Type	Description	Label
side_chain_id	*int32*	The side chain id to adjust.	
indexing_fee	*int64*	The new price of indexing fee.	

acs7.ChainIdAndHeightDict

Field	Type	Description	Label
id_height_dict	*ChainIdAndHeightDict.IdHeightDictEntry*	A collection of chain ids and heights, where the key is the chain id and the value is the height.	repeated

acs7.ChainIdAndHeightDict.IdHeightDictEntry

Field	Type	Description	Label
key	*int32*		
value	*int64*		

acs7.ChainInitializationConsensusInfo

Field	Type	Description	Label
initial_consensus_data	*bytes*	Initial consensus data.	

acs7.ChainInitializationData

Field	Type	Description	Label
chain_id	*int32*	The id of side chain.	
creator	*aelf.Address*	The side chain creator.	
creation_timestamp	*google.protobuf.Timestamp*	The timestamp for side chain creation.	
creation_height_on_parent_chain	*int64*	The height of side chain creation on parent chain.	
chain_creator_privilege_preserved	*bool*	Creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.	
parent_chain_token_contract_address	*aelf.Address*	Parent chain token contract address.	
chain_initialization_consensus_info	*ChainInitializationConsensusInfo*	Initial consensus information.	
native_token_info_data	*bytes*	The native token info.	
resource_token_info	*ResourceTokenInfo*	The resource token information.	
chain_primary_token_info	*ChainPrimaryTokenInfo*	The chain primary token information.	

acs7.ChainPrimaryTokenInfo

Field	Type	Description	Label
chain_primary_token_data	*bytes*	The side chain primary token data.	
side_chain_token_initial_issue_list	*SideChainTokenInitialIssue*	The side chain primary token initial issue list.	repeated

acs7.CreateSideChainInput

Field	Type	Description	Label
side_chain_creation_request	*SideChainCreationRequest*	The request information of the side chain creation.	
proposer	*aelf.Address*	The proposer of the side chain creation.	

acs7.CrossChainBlockData

Field	Type	Description	Label
side_chain_block_data_list	*SideChainBlockData*	The side chain block data list to index.	repeated
parent_chain_block_data_list	*ParentChainBlockData*	The parent chain block data list to index.	repeated

acs7.CrossChainExtraData

Field	Type	Description	Label
transaction_status_merkle_tree_root	*aelf.Hash*	Merkle tree root of side chain block transaction status root.	

acs7.CrossChainIndexingDataProposedEvent

Field	Type	Description	Label
proposed_cross_chain_data	*CrossChainBlockData*	Proposed cross chain data to be indexed.	
proposal_id	*aelf.Hash*	The proposal id.	

acs7.CrossChainMerkleProofContext

Field	Type	Description	Label
bound_parent_chain_height	*int64*	The height of parent chain bound up with side chain.	
merkle_path_from_parent_chain	*aelf.MerklePath*	The merkle path generated from parent chain.	

acs7.IndexedParentChainBlockData

Field	Type	Description	Label
local_chain_height	*int64*	The height of the local chain when indexing the parent chain.	
parent_chain_block_data_list	*ParentChainBlockData*	Parent chain block data.	repeated

acs7.IndexedSideChainBlockData

Field	Type	Description	Label
side_chain_block_data_list	*SideChainBlockData*	Side chain block data.	repeated

acs7.ParentChainBlockData

Field	Type	Description	Label
height	*int64*	The height of parent chain.	
cross_chain_extra_data	*CrossChainExtraData*	The merkle tree root computing from side chain roots.	
chain_id	*int32*	The parent chain id.	
transaction_status_merkle_tree_root	*aelf.Hash*	The merkle tree root computing from transactions status in parent chain block.	
indexed_merkle_path	*ParentChainBlockData.IndexedMerklePathEntry*	Indexed block height from side chain and merkle path for this side chain block	repeated
extra_data	*ParentChainBlockData.ExtraDataEntry*	Extra data map.	repeated

acs7.ParentChainBlockData.ExtraDataEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

acs7.ParentChainBlockData.IndexedMerklePathEntry

Field	Type	Description	Label
key	*int64*		
value	*aelf.MerklePath*		

acs7.RechargeInput

Field	Type	Description	Label
chain_id	*int32*	The chain id to recharge.	
amount	*int64*	The amount to recharge.	

acs7.ReleaseCrossChainIndexingProposalInput

Field	Type	Description	Label
chain_id_list	*int32*	List of chain ids to release.	repeated

acs7.ReleaseSideChainCreationInput

Field	Type	Description	Label
proposal_id	*aelf.Hash*	The proposal id of side chain creation.	

acs7.ResourceTokenInfo

Field	Type	Description	Label
resource_token_list_data	*bytes*	The resource token information.	
initial_resource_amount	*ResourceTokenInfo.InitialResourceAmountEntry*	The initial resource token amount.	repeated

acs7.ResourceTokenInfo.InitialResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

acs7.SideChainBlockData

Field	Type	Description	Label
height	*int64*	The height of side chain block.	
block_header_hash	*aelf.Hash*	The hash of side chain block.	
transaction_status_merkle_tree_root	*aelf.Hash*	The merkle tree root computing from transactions status in side chain block.	
chain_id	*int32*	The id of side chain.	

acs7.SideChainBlockDataIndexed

acs7.SideChainCreationRequest

Field	Type	Description	Label
indexing_price	*int64*	The cross chain indexing price.	
locked_token_amount	*int64*	Initial locked balance for a new side chain.	
is_privilege_preserved	*bool*	Creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.	
side_chain_token_creation_request	*SideChainTokenCreationRequest*	Side chain token information.	
side_chain_token_initial_issue_list	*SideChainTokenInitialIssue*	A list of accounts and amounts that will be issued when the chain starts.	repeated
initial_resource_amount	*SideChainCreationRequest.InitialResourceAmountEntry*	The initial rent resources.	repeated

acs7.SideChainCreationRequest.InitialResourceAmountEntry

Field	Type	Description	Label
key	*string*		
value	*int32*		

acs7.SideChainIndexingInformation

Field	Type	Description	Label
chain_id	*int32*	The side chain id.	
indexed_height	*int64*	The indexed height.	

acs7.SideChainIndexingInformationList

Field	Type	Description	Label
indexing_information_list	*SideChainIndexingInformation*	A list contains indexing information of side chains.	repeated

acs7.SideChainTokenCreationRequest

Field	Type	Description	Label
side_chain_token_symbol	*string*	Token symbol of the side chain to be created	
side_chain_token_name	*string*	Token name of the side chain to be created	
side_chain_token_total_supply	*int64*	Token total supply of the side chain to be created	
side_chain_token_decimals	*int32*	Token decimals of the side chain to be created	

acs7.SideChainTokenInitialIssue

Field	Type	Description	Label
address	*aelf.Address*	The account that will be issued.	
amount	*int64*	The amount that will be issued.	

acs7.VerifyTransactionInput

Field	Type	Description	Label
transaction_id	*aelf.Hash*	The cross chain transaction id to verify.	
path	*aelf.MerklePath*	The merkle path of the transaction.	
parent_chain_height	*int64*	The height of parent chain that indexing this transaction.	
verified_chain_id	*int32*	The chain if to verify.	

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.8.2 Example

ACS7 declares methods for the scenes about cross chain. AElf provides the implementation for ACS7, `CrossChainContract`. You can refer to the implementation of the *Cross chain contract api*.

21.9 ACS8 - Transaction Resource Token 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.

21.9.1 Interface

Only one method is defined in the acs8.proto file:

Methods

Method Name	Request Type	Response Type	Description
BuyResourceToken	*acs8.BuyResourceTokenInput*	*google.protobuf.Empty*	Buy 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).

Types

acs8.BuyResourceTokenInput

Field	Type	Description	Label
symbol	*string*	The symbol token you want to buy.	
amount	*int64*	The amount you want to buy.	
pay_limit	*int64*	Limit of cost. If the token required for buy exceeds this value, the buy will be abandoned. And 0 is no limit.	

21.9.2 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` transaction 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));
```

(continues on next page)

(continued from previous page)

```
        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();
}
```

21.10 ACS9 - Contract profit dividend standard

On the AElf's side chain, the contract needs to declare where its profits are going, and implement ACS9.

21.10.1 Interface

ACS9 contains an method which does not have to be implemented:

Methods

Method Name	Request Type	Response Type	Description
TakeContractProfits	*acs9.TakeContractProfitsInput*	*google.protobuf.Empty*	Used for the developer to collect the profits from the contract and the profits will be distributed in this method.
GetProfitConfig	*google.protobuf.Empty*	*acs9.ProfitConfig*	Query the config of profit.
GetProfitsAmount	*google.protobuf.Empty*	*acs9.ProfitsMap*	Query the profits of the contract so far.

Types

acs9.ProfitConfig

Field	Type	Description	Label
donation_parts_per_hundred	*int32*	The portion of the profit that will be donated to the dividend pool each time the developer receives the profit.	
profits_token_symbol_list	*string*	The profit token symbol list.	repeated
staking_token_symbol	*string*	The token symbol that the user can lock them to claim the profit.	

acs9.ProfitsMap

Field	Type	Description	Label
value	*ProfitsMap.ValueEntry*	The profits, token symbol -> amount.	repeated

acs9.ProfitsMap.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs9.TakeContractProfitsInput

Field	Type	Description	Label
symbol	*string*	The token symbol to take.	
amount	*int64*	The amount to take.	

21.10.2 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,
        IsProfitable = true,
        TotalSupply = ACS9DemoContractConstants.TotalSupply,
        LockWhiteList =
        {
            lockWhiteList
        }
    });
    State.TokenContract.Issue.Send(new IssueInput
    {
        To = profitReceiver,
```

(continues on next page)

(continued from previous page)

```
        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,
        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
    });
    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;
```

(continues on next page)

(continued from previous page)

```
    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
    });
    // 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,
```

(continues on next page)

(continued from previous page)

```
        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;
}
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;
}
```

21.10.3 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>
        {
```

(continues on next page)

(continued from previous page)

```
            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.
```

(continues on next page)

(continued from previous page)

```
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",
```

(continues on next page)

(continued from previous page)

```
    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.
{
    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",
```

(continues on next page)

(continued from previous page)

```
    Owner = UserAddresses[0]
});
resultBalance.Balance.ShouldBe(beforeBalance.Balance + 57_00000000);
```

21.11 ACS10 - Dividend Pool Standard

ACS10 is used to construct a dividend pool in the contract.

21.11.1 Interface

To construct a dividend pool, you can implement the following interfaces optionally:

Methods

Method Name	Request Type	Response Type	Description
Donate	*acs10.DonateInput*	*google.protobuf.Empty*	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.
Release	*acs10.ReleaseInput*	*google.protobuf.Empty*	Release dividend pool according the period number.
SetSymbolList	*acs10.SymbolList*	*google.protobuf.Empty*	Set the token symbols dividend pool supports.
GetSymbolList	*google.protobuf.Empty*	*acs10.SymbolList*	Query the token symbols dividend pool supports.
GetUndistributedDividends	*google.protobuf.Empty*	*acs10.Dividends*	Query the balance of undistributed tokens whose symbols are included in the symbol list.
GetDividends	*google.protobuf.Int64Value*	*acs10.Dividends*	Query the dividend information according to the height.

Types

acs10.Dividends

Field	Type	Description	Label
value	*Dividends.ValueEntry*	The dividends, symbol -> amount.	repeated

acs10.Dividends.ValueEntry

Field	Type	Description	Label
key	*string*		
value	*int64*		

acs10.DonateInput

Field	Type	Description	Label
symbol	*string*	The token symbol to donate.	
amount	*int64*	The amount to donate.	

acs10.DonationReceived

Field	Type	Description	Label
from	*aelf.Address*	The address of donors.	
pool_contract	*aelf.Address*	The address of dividend pool.	
symbol	*string*	The token symbol Donated.	
amount	*int64*	The amount Donated.	

acs10.ReleaseInput

Field	Type	Description	Label
period_number	*int64*	The period number to release.	

acs10.SymbolList

Field	Type	Description	Label
value	*string*	The token symbol list.	repeated

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.11.2 Usage

ACS10 only unifies the standard interface of the dividend pool, which does not interact with the AElf chain.

21.11.3 Implementation

With the Profit contract

A Profit Scheme can be created using the `CreateScheme` method of `Profit contract`:

```
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 included 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
```

(continues on next page)

(continued from previous page)

```
});
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,
        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;
```

(continues on next page)

(continued from previous page)

```
    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 `GetUndistributedDividends` is the same as described in the previous section, and it returns the balance:

```
public override Dividends GetUndistributedDividends(Empty input)
{
    var scheme = GetDividendPoolScheme();
```

(continues on next page)

(continued from previous page)

```
    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.

21.11.4 Test

The dividend pool, for example, is tested in two ways with the `TokenHolder 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
```

(continues on next page)

(continued from previous page)

```
{
    Owner = address,
    Symbol = "ELF"
});
balanceAfterClaimForProfits.Balance.ShouldBe(balanceBeforeClaimForProfits.Balance +
↪amount);
```

21.11.5 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 `Consensus contract` implementing ACS10 is on the side chain.

21.12 ACS11 - Cross Chain Consensus Standard

ACS11 is used to customize consensus mechanisms for cross chain.

21.12.1 Interface

The contract inherited from ACS11 need implement the following interfaces:

Methods

Method Name	Request Type	Response Type	Description
UpdateInformationFromCrossChain	*google.protobuf.BytesValue*	*google.protobuf.Empty*	Update the consensus information of the side chain.
GetChainInitializationInformation	*google.protobuf.BytesValue*	*google.protobuf.BytesValue*	Get the current miner list and consensus round information.
CheckCrossChainIndexingPermission	*aelf.Address*	*google.protobuf.BoolValue*	Verify that the input address is the current miner.

Types

aelf.Address

Field	Type	Description	Label
value	*bytes*		

aelf.BinaryMerkleTree

Field	Type	Description	Label
nodes	*Hash*	The leaf nodes.	repeated
root	*Hash*	The root node hash.	
leaf_count	*int32*	The count of leaf node.	

aelf.Hash

Field	Type	Description	Label
value	*bytes*		

aelf.LogEvent

Field	Type	Description	Label
address	*Address*	The contract address.	
name	*string*	The name of the log event.	
indexed	*bytes*	The indexed data, used to calculate bloom.	repeated
non_indexed	*bytes*	The non indexed data.	

aelf.MerklePath

Field	Type	Description	Label
merkle_path_nodes	*MerklePathNode*	The merkle path nodes.	repeated

aelf.MerklePathNode

Field	Type	Description	Label
hash	*Hash*	The node hash.	
is_left_child_node	*bool*	Whether it is a left child node.	

aelf.SInt32Value

Field	Type	Description	Label
value	*sint32*		

aelf.SInt64Value

Field	Type	Description	Label
value	*sint64*		

aelf.ScopedStatePath

Field	Type	Description	Label
address	*Address*	The scope address, which will be the contract address.	
path	*StatePath*	The path of contract state.	

aelf.SmartContractRegistration

Field	Type	Description	Label
category	*sint32*	The category of contract code(0: C#).	
code	*bytes*	The byte array of the contract code.	
code_hash	*Hash*	The hash of the contract code.	
is_system_contract	*bool*	Whether it is a system contract.	
version	*int32*	The version of the current contract.	

aelf.StatePath

Field	Type	Description	Label
parts	*string*	The partial path of the state path.	repeated

aelf.Transaction

Field	Type	Description	Label
from	*Address*	The address of the sender of the transaction.	
to	*Address*	The address of the contract when calling a contract.	
ref_block_number	*int64*	The height of the referenced block hash.	
ref_block_prefix	*bytes*	The first four bytes of the referenced block hash.	
method_name	*string*	The name of a method in the smart contract at the To address.	
params	*bytes*	The parameters to pass to the smart contract method.	
signature	*bytes*	When signing a transaction it's actually a subset of the fields: from/to and the target method as well as the parameter that were given. It also contains the reference block number and prefix.	

aelf.TransactionExecutingStateSet

Field	Type	Description	Label
writes	*TransactionExecutingStateSet.WritesEntry*	The changed states.	repeated
reads	*TransactionExecutingStateSet.ReadsEntry*	The read states.	repeated
deletes	*TransactionExecutingStateSet.DeletesEntry*	The deleted states.	repeated

aelf.TransactionExecutingStateSet.DeletesEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.ReadsEntry

Field	Type	Description	Label
key	*string*		
value	*bool*		

aelf.TransactionExecutingStateSet.WritesEntry

Field	Type	Description	Label
key	*string*		
value	*bytes*		

aelf.TransactionResult

Field	Type	Description	Label
transaction_id	*Hash*	The transaction id.	
status	*TransactionResultStatus*	The transaction result status.	
logs	*LogEvent*	The log events.	repeated
bloom	*bytes*	Bloom filter for transaction logs. A transaction log event can be defined in the contract and stored in the bloom filter after the transaction is executed. Through this filter, we can quickly search for and determine whether a log exists in the transaction result.	
return_value	*bytes*	The return value of the transaction execution.	
block_number	*int64*	The height of the block hat packages the transaction.	
block_hash	*Hash*	The hash of the block hat packages the transaction.	
error	*string*	Failed execution error message.	

aelf.TransactionResultStatus

Name	Number	Description
NOT_EXISTED	0	The execution result of the transaction does not exist.
PENDING	1	The transaction is in the transaction pool waiting to be packaged.
FAILED	2	Transaction execution failed.
MINED	3	The transaction was successfully executed and successfully packaged into a block.
CONFLICT	4	When executed in parallel, there are conflicts with other transactions.
PENDING_VALIDATION	5	The transaction is waiting for validation.
NODE_VALIDATION_FAILED	6	Transaction validation failed.

21.12.2 Example

ACS11 declares methods for the scenes about customize consensus mechanisms for cross chain. AElf provides the implementation for ACS11, `AEDPoS Contract`. You can refer to the implementation of the *AEDPoS contract api*.

CHAPTER 22

Command line interface

22.1 Introduction to the CLI

The **aelf-command** tool is a CLI tool built for interacting with an AElf node. This section will walk you through some of the most commonly used features and show you how to install the tool.

22.1.1 Features

- Get or Set common configs, `endpoint`, `account`, `datadir`, `password`.
- For new users who are not familiar with the CLI parameters, any missing parameters will be asked in a prompting way.
- Create a new `account`.
- Load an account from a given `private key` or `mnemonic`.
- Show `wallet` details which include private key, address, public key and mnemonic.
- Encrypt account info into `keyStore` format and save to file.
- Get current `Best Height` of the chain.
- Get `block info` by a given `height` or `block hash`.
- Get `transaction result` by a given `transaction id`.
- Send a `transaction` or call a `read-only method` on a smart `contract`.
- Deploy a smart `contract`.
- Open a `REPL` for using `JavaScript` to interact with the chain.
- Friendly interactions, beautify with chalk & ora.
- Get current chain status.
- Create a proposal on any contract method.

- Deserialize the result returned by executing a transaction.
- Start a socket.io server for supplying services for dApps.

22.1.2 Install aelf-command

```
npm i aelf-command -g
```

22.1.3 Using aelf-command

First Step

You need to create a new account or load a account by a `private key` or `mnemonic` you already have.

- Create a new wallet

```
$ aelf-command create
Your wallet info is :
Mnemonic            : great mushroom loan crisp ... door juice embrace
Private Key         : e038eea7e151eb451ba2901f7...b08ba5b76d8f288
Public Key          : 0478903d96aa2c8c0...
↪6a3e7d810cacd136117ea7b13d2c9337e1ec88288111955b76ea
Address             : 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✔ Save account info into a file? ... no / yes
✔ Enter a password ... ********
✔ Confirm password ... ********
✔
Account info has been saved to "/Users/young/.local/share/aelf/keys/
↪2Ue31YTuB5Szy7cnr...Gi5uMQBYarYUR5oGin1sys6H.json"
```

- Load wallet from private key

```
$ aelf-command load e038eea7e151eb451ba2901f7...b08ba5b76d8f288
Your wallet info is :
Private Key         : e038eea7e151eb451ba2901f7...b08ba5b76d8f288
Public Key          : 0478903d96aa2c8c0...
↪6a3e7d810cacd136117ea7b13d2c9337e1ec88288111955b76ea
Address             : 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✔ Save account info into a file?
✔ Enter a password ... ********
✔ Confirm password ... ********
✔
Account info has been saved to "/Users/young/.local/share/aelf/keys/
↪2Ue31YTuB5Szy7cnr...Gi5uMQBYarYUR5oGin1sys6H.json"
```

- show wallet info you already have

```
$ aelf-command wallet -a 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
Your wallet info is :
Private Key         : e038eea7e151eb451ba2901f7...b08ba5b76d8f288
Public Key          : 0478903d96aa2c8c0...
↪6a3e7d810cacd136117ea7b13d2c9337e1ec88288111955b76ea
Address             : 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
```

Here you can get the account info and decide whether to encrypt account info and save into a file.

Examples:

```
$ aelf-command console -a 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✓ Enter the the URI of an AElf node: http://127.0.0.1:8000
✓ Enter the password you typed when creating a wallet ... ********
✓ Succeed!
Welcome to aelf interactive console. Ctrl + C to terminate the program. Double tap␣
↪Tab to list objects



      NAME        | DESCRIPTION
      AElf        | imported from aelf-sdk
      aelf        | the instance of an aelf-sdk, connect to
                  | http://127.0.0.1:8000
      _account    | the instance of an AElf wallet, address
                  | is
                  | 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR...
                  | 5oGin1sys6H
```

Any missed parameters you did not give in CLI parameters will be asked in a prompting way

```
$ aelf-command console
✓ Enter the the URI of an AElf node: http://127.0.0.1:8000
✓ Enter a valid wallet address, if you don\'t have, create one by aelf-command create␣
↪... 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✓ Enter the password you typed when creating a wallet ... ********
✓ Succeed!
Welcome to aelf interactive console. Ctrl + C to terminate the program. Double tap␣
↪Tab to list objects



      NAME        | DESCRIPTION
      AElf        | imported from aelf-sdk
      aelf        | the instance of an aelf-sdk, connect to
                  | http://13.231.179.27:8000
      _account    | the instance of an AElf wallet, address
                  | is
                  | 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR...
                  | 5oGin1sys6H
```

Help

Type

```
$ aelf-command -h
Usage: aelf-command [command] [options]

Options:
  -v, --version                                         output the version number
  -e, --endpoint <URI>                                  The URI of an AElf node.␣
↪Eg: http://127.0.0.1:8000
```

(continues on next page)

(continued from previous page)

```
  -a, --account <account>                                      The address of AElf wallet
  -p, --password <password>                                    The password of encrypted␣
↪keyStore
  -d, --datadir <directory>                                    The directory that␣
↪contains the AElf related files. Defaults to {home}/.local/share/aelf
  -h, --help                                                   output usage information

Commands:
  call [contract-address|contract-name] [method] [params]     Call a read-only method␣
↪on a contract.
  send [contract-address|contract-name] [method] [params]     Execute a method on a␣
↪contract.
  get-blk-height                                               Get the current block␣
↪height of specified chain
  get-chain-status                                             Get the current chain␣
↪status
  get-blk-info [height|block-hash] [include-txs]               Get a block info
  get-tx-result [tx-id]                                        Get a transaction result
  console                                                      Open a node REPL
  create [options] [save-to-file]                              Create a new account
  wallet                                                       Show wallet details␣
↪which include private key, address, public key and mnemonic
  load [private-key|mnemonic] [save-to-file]                   Load wallet from a␣
↪private key or mnemonic
  proposal [proposal-contract] [organization] [expired-time]  Send a proposal to an␣
↪origination with a specific contract method
  deploy [category] [code-path]                                Deprecated! Please use ␣
↪`aelf-command send` , check details in aelf-command `README.md`
  config <flag> [key] [value]                                  Get, set, delete or␣
↪list aelf-command config
  event [tx-id]                                                Deserialize the result␣
↪returned by executing a transaction
  dapp-server [options]                                        Start a dAPP SOCKET.IO␣
↪server
```

in your terminal and get useful information.

Any sub-commands such as `call`, you can get `help` by typing this

```
$ aelf-command call -h
Usage: aelf-command call [options] [contract-address|contract-name] [method] [params]

Call a read-only method on a contract.

Options:
  -h, --help  output usage information

Examples:

aelf-command call <contractName|contractAddress> <method> <params>
aelf-command call <contractName|contractAddress> <method>
aelf-command call <contractName|contractAddress>
aelf-command call

$ aelf-command console -h
Usage: aelf-command console [options]
```

(continues on next page)

(continued from previous page)

```
Open a node REPL

Options:
  -h, --help  output usage information

Examples:

aelf-command console
...
```

22.2 Commands

22.2.1 Common options

- `datadir`: The directory that contains `aelf-command` files, such as `encrypted account info keyStore` files. Default to be `{home}/.local/share/aelf`
- `endpoint`: The endpoint for the RPC service.
- `account`: The account to be used to interact with the blockchain `endpoint`.
- `password`: The password for unlocking the given `account`.

You can specified options above in several ways, and the priority is in the order of low to high.

1. `export` variables in shell.

   ```
   # This is datadir
   $ export AELF_CLI_DATADIR=/Users/{you}/.local/share/aelf
   # This is endpoint
   $ export AELF_CLI_ENDPOINT=http://127.0.0.1:8000
   # This is account
   $ export AELF_CLI_ACCOUNT=2Ue31YTuB5Szy7c...gtGi5uMQBYarYUR5oGin1sys6H
   ```

2. `aelf-command` global `.aelfrc` config file

   The global config file is stored in the `<datadir>/.aelfrc` file, you can read the config file, but better not modify it by yourself.

   Modify this config file by `aelf-command config`.

   - `set`: set and save config in the file, remember just set the `datadir`, `endpoint`, `account`, `password` four keys.

     ```
     $ aelf-command config set endpoint http://127.0.0.1:8000
     ✔ Succeed!

     $ aelf-command config -h
     Usage: aelf-command config [options] <flag> [key] [value]

     get, set, delete or list aelf-command config

     Options:
       -h, --help  output usage information

     Examples:
     ```

(continues on next page)

(continued from previous page)

```
aelf-command config get <key>
aelf-command config set <key> <value>
aelf-command config delete <key>
aelf-command config list
```

- `get`: get the value of given `key` from global `.aelfrc` file

```
$ aelf-command config get endpoint
http://127.0.0.1:8000
```

- `delete`: delete the `<key, value>` from global `.aelfrc` file by a given key

```
$ aelf-command config delete endpoint
✓ Succeed!
```

- `list`: get the list of all configs stored in global `.aelfrc` file

```
$ aelf-command config list
endpoint=http://127.0.0.1:8000
password=password
```

Remember `config` command only can be used to modify the global `.aelfrc` file for now, more usages such as modify working directory will be implemented in later.

3. `aelf-command` working directory `.aelfrc` file

   The current working directory of `aelf-command` can have a file named `.aelfrc` and store configs, the format of this file is like global `.aelfrc` file:

```
endpoint http://127.0.0.1:8000
password yourpassword
```

   each line is `<key, value>` config and a whitespace is needed to separate them.

4. `aelf-command` options.

   You can give common options by passing them in CLI parameters.

```
aelf-command console -a sadaf -p password -e http://127.0.0.1:8000
```

   Notice the priority, the options given in higher priority will overwrite the lower priority.

22.2.2 create - Create a new account

This command will create a new account.

```
$ aelf-command create -h
Usage: aelf-command create [options] [save-to-file]

create a new account

Options:
  -c, --cipher [cipher]  Which cipher algorithm to use, default to be aes-128-ctr
  -h, --help             output usage information
```

(continues on next page)

(continued from previous page)

```
Examples:

aelf-command create <save-to-file>
aelf-command create
```

Example:

- Specify the cipher way to encrypt account info by passing option `-c [cipher]`, such as:

```
aelf-command create -c aes-128-cbc
```

22.2.3 load - Load an account by a given `private key` or `mnemonic`

This command allow you load an account from backup.

```
# load from mnemonic
$ aelf-command load 'great mushroom loan crisp ... door juice embrace'
# load from private key
$ aelf-command load 'e038eea7e151eb451ba2901f7...b08ba5b76d8f288'
# load from prompting
$ aelf-command load
? Enter a private key or mnemonic › e038eea7e151eb451ba2901f7...b08ba5b76d8f288
...
```

22.2.4 wallet - Show wallet details which include `private key`, `address`, `public key` and `mnemonic`

This command allows you to print wallet info.

```
$ aelf-command wallet -a C91b1SF5mMbenHZTfdfbJSkJcK7HMjeiuw...8qYjGsESanXR
AElf [Info]: Private Key         : 97ca9fbece296231f26bee0e493500810f...
↪cbd984f69a8dc22ec9ec89ebb00
AElf [Info]: Public Key          : 04c30dd0c3b5abfc85a11b15dabd0de926...
↪74fe04e92eaebf2e4fef6445d9b9b11efe6f4b70c8e86644b72621f9987dc00bb1eab44a9bd7512ea53f93937a5d0
AElf [Info]: Address             : C91b1SF5mMbenHZTfdfbJSkJcK7HMjeiuw...8qYjGsESanXR
```

22.2.5 proposal - Create a proposal

There are three kinds of proposal contracts in AElf:

- `AElf.ContractNames.Parliament`
- `AElf.ContractNames.Referendum`
- `AElf.ContractNames.Association`

depending on your needs you can choose one and create a proposal.

- Get an organization address or create one

Get the default organization's address with the parliament contract (`AElf.ContractNames.Parliament`):

```
$ aelf-command call AElf.ContractNames.Parliament GetDefaultOrganizationAddress
✔ Fetching contract successfully!
✔ Calling method successfully!
AElf [Info]:
Result:
"BkcXRkykRC2etHp9hgFfbw2ec1edx7ERBxYtbC97z3Q2bNCwc"
✔ Succeed!
```

BkcXRkykRC2etHp9hgFfbw2ec1edx7ERBxYtbC97z3Q2bNCwc is the default organization address.

The default organization is an organization that contains all miners; every proposal under AElf.ContractNames.Parliament can only be released when it has got over 2/3 miners approval.

Create an organization with the Referendum contract (AElf.ContractNames.Referendum):

```
$ aelf-command send AElf.ContractNames.Referendum
✔ Fetching contract successfully!
? Pick up a contract method: CreateOrganization

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional parameters:
? Enter the required param <tokenSymbol>: ELF
? Enter the required param <proposalReleaseThreshold.minimalApprovalThreshold>: 666
? Enter the required param <proposalReleaseThreshold.maximalRejectionThreshold>: 666
? Enter the required param <proposalReleaseThreshold.maximalAbstentionThreshold>: 666
? Enter the required param <proposalReleaseThreshold.minimalVoteThreshold>: 666
? Enter the required param <proposerWhiteList.proposers>: [
↪"2hxkDg6Pd2d4yU1A16PTZVMMrEDYEPR8oQojMDwWdax5LsBaxX"]
The params you entered is:
{
  "tokenSymbol": "ELF",
  "proposalReleaseThreshold": {
    "minimalApprovalThreshold": 666,
    "maximalRejectionThreshold": 666,
    "maximalAbstentionThreshold": 666,
    "minimalVoteThreshold": 666
  },
  "proposerWhiteList": {
    "proposers": [
      "2hxkDg6Pd2d4yU1A16PTZVMMrEDYEPR8oQojMDwWdax5LsBaxX"
    ]
  }
}
✔ Succeed!
AElf [Info]:
Result:
{
  "TransactionId": "273285c7e8825a0af5291dd5d9295f746f2bb079b30f915422564de7a64fc874"
}
✔ Succeed!
```

- Create a proposal

```
$ aelf-command proposal
? Pick up a contract name to create a proposal: AElf.ContractNames.Parliament
? Enter an organization address: BkcXRkykRC2etHp9hgFfbw2ec1edx7ERBxYtbC97z3Q2bNCwc
```

(continues on next page)

(continued from previous page)

```
? Select the expired time for this proposal: 2022/09/23 22:06
? Optional, input an URL for proposal description:
? Enter a contract address or name: AElf.ContractNames.Token
✓ Fetching contract successfully!
? Pick up a contract method: Transfer

If you need to pass file contents to the contractMethod, you can enter the relative␣
↪or absolute path of the file instead

Enter required params one by one:
? Enter the required param <to>: 2hxkDg6Pd2d4yU1A16PTZVMMrEDYEPR8oQojMDwWdax5LsBaxX
? Enter the required param <symbol>: ELF
? Enter the required param <amount>: 100000000
? Enter the required param <memo>: test
AElf [Info]:
 { TransactionId:
   '09c8c824d2e3aea1d6cd15b7bb6cefe4e236c5b818d6a01d4f7ca0b60fe99535' }
✓ loading proposal id...
AElf [Info]: Proposal id:
↪"bafe83ca4ec5b2a2f1e8016d09b21362c9345954a014379375f1a90b7afb43fb".
✓ Succeed!
```

You can get the proposal id, then get the proposal's status.

- Get proposal status

```
$ aelf-command call AElf.ContractNames.Parliament GetProposal␣
↪bafe83ca4ec5b2a2f1e8016d09b21362c9345954a014379375f1a90b7afb43fb
{
  ...
  "expiredTime": {
    "seconds": "1663942010",
    "nanos": 496000
  },
  "organizationAddress": "BkcXRkykRC2etHp9hgFfbw2ec1edx7ERBxYtbC97z3Q2bNCwc",
  "proposer": "2tj7Ea67fuQfVAtQZ3WBmTv7AAJ8S9D2L4g6PpRRJei6JXk7RG",
  "toBeReleased": false
}
✓ Succeed!
```

`toBeReleased` indicates whether you can release this proposal. By default, a proposal needs over 2/3 BP nodes approval.

- Release a proposal

You can release a proposal when it got approved.

```
$ aelf-command send AElf.ContractNames.Parliament Release␣
↪bafe83ca4ec5b2a2f1e8016d09b21362c9345954a014379375f1a90b7afb43fb
AElf [Info]:
 { TransactionId:
   '09c8c824d2e3aea1d...cefe4e236c5b818d6a01d4f7ca0b60fe99535' }
```

Get the transaction result

```
$ aelf-command get-tx-result 09c8c824d2e3aea1d...cefe4e236c5b818d6a01d4f7ca0b60fe99535
AElf [Info]: {
```

(continues on next page)

(continued from previous page)

```
  "TransactionId": "09c8c824d2e3aea1d...cefe4e236c5b818d6a01d4f7ca0b60fe99535",
  "Status": "MINED",
  "Logs": [
    {
    "Address": "25CecrU94dmMdbhC3LWMKxtoaL4Wv8PChGvVJM6PxkHAyvXEhB",
    "Name": "Transferred",
    "Indexed": [
      "CiIKIJTPGZ24g4eHwSVNLit8jgjFJeeYCEEYLDpFiCeCT0Bf",
      "EiIKIO0jJRxjHdRQmUTby8klRVSqYpwhOyUsnXYV3IrQg8N1",
      "GgNFTEY="
    ],
    "NonIndexed": "IICgt4fpBSomVC00MzFkMjc0Yi0zNWJjLTRjYzgtOGExZC1iODhhZTgxYzU2Zjc="
    }
  ],
  "Bloom":
↪"AAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
  "BlockNumber": 28411,
  "BlockHash": "fa22e4eddff12a728895a608db99d40a4b21894f7c07df1a4fa8f0625eb914a2",
  "Transaction": {
    "From": "2tj7Ea67fuQfVAtQZ3WBmTv7AAJ8S9D2L4g6PpRRJei6JXk7RG",
    "To": "29RDBXTqwnpWPSPHGatYsQXW2E17YrQUCj7QhcEZDnhPb6ThHW",
    "RefBlockNumber": 28410,
    "RefBlockPrefix": "0P+eTw==",
    "MethodName": "Release",
    "Params": "\"ad868c1e0d74127dd746ccdf3443a09459c55cf07d247df053ddf718df258c86\"",
    "Signature": "DQcv55EBWunEFPXAbqZG20OLO5T0Sq/s0A+/
↪iuwv1TdQqIV4318HrqFLsGpx9m3+sp5mzhAnMlrG7CSxM6EuIgA="
  },
  "ReturnValue": "",
  "Error": null
}
```

If you want to call a contract method by creating a proposal and released it, the released transaction result could be confusing, you can use another `aelf-command` sub-command to get the readable result;

Take the example above which has transferred token by proposal, transferred result can be viewed by decoding the `Logs` field in the transaction result. Use `aelf-command event` to decode the results.

Pass the transaction id as a parameter:

```
$ aelf-command event 09c8c824d2e3aea1d...cefe4e236c5b818d6a01d4f7ca0b60fe99535
[Info]:
The results returned by
Transaction: 09c8c824d2e3aea1d...cefe4e236c5b818d6a01d4f7ca0b60fe99535 is:
[
  {
    "Address": "25CecrU94dmMdbhC3LWMKxtoaL4Wv8PChGvVJM6PxkHAyvXEhB",
    "Name": "Transferred",
    "Indexed": [
      "CiIKIJTPGZ24g4eHwSVNLit8jgjFJeeYCEEYLDpFiCeCT0Bf",
      "EiIKIO0jJRxjHdRQmUTby8klRVSqYpwhOyUsnXYV3IrQg8N1",
      "GgNFTEY="
    ],
    "NonIndexed": "IICgt4fpBSomVC00MzFkMjc0Yi0zNWJjLTRjYzgtOGExZC1iODhhZTgxYzU2Zjc=",
    "Result": {
      "from": "28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK",
```

(continues on next page)

(continued from previous page)

```
      "to": "2oSMWm1tjRqVdfmrdL8dgrRvhWu1FP8wcZidjS6wPbuoVtxhEz",
      "symbol": "ELF",
      "amount": "200000000000",
      "memo": "T-431d274b-35bc-4cc8-8a1d-b88ae81c56f7"
    }
  }

]
```

The `Result` field is the decoded result.

For more details, check the descriptions of *aelf-command event*.

22.2.6 deploy - Deploy a smart contract

This command has been deprecated, use `aelf-command send` or `aelf-command proposal` instead

Examples:

1. Use Genesis Contract to deploy a new smart contract

```
$ aelf-command get-chain-status
✔ Succeed
{
  "ChainId": "AELF",
  "Branches": {
    "41a8a1ebf037197b7e2f10a67d81f741d46a6af41775bcc4e52ab855c58c4375": 8681551,
    "ed4012c21a2fbf810db52e9869ef6a3fb0629b36d23c9be2e3692a24703b3112": 8681597,
    "13476b902ef137ed63a4b52b2902bb2b2fa5dbe7c256fa326c024a73dc63bcb3": 8681610
  },
  "NotLinkedBlocks": {},
  "LongestChainHeight": 8681610,
  "LongestChainHash":
↪"13476b902ef137ed63a4b52b2902bb2b2fa5dbe7c256fa326c024a73dc63bcb3",
  "GenesisBlockHash":
↪"cd5ce1bfa0cd97a1dc34f735c57bea2fcb9d88fc8f76bece2592fe7d82d5660c",
  "GenesisContractAddress": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
  "LastIrreversibleBlockHash":
↪"4ab84cdfe0723b191eedcf4d2ca86b0f64e57105e61486c21d98d562b14f2ab0",
  "LastIrreversibleBlockHeight": 8681483,
  "BestChainHash":
↪"0dbc2176aded950020577552c92c82e66504ea109d4d6588887502251b7e932b",
  "BestChainHeight": 8681609
}

# use GenesisContractAddress as a parameter of aelf-command send
# use contract method `DeploySmartContract` if the chain you are connecting to␣
↪requires no limit of authority
$ aelf-command send 2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8␣
↪DeploySmartContract
✔ Fetching contract successfully!

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
```

(continues on next page)

(continued from previous page)

```
? Enter the required param <category>: 0
? Enter the required param <code>: /Users/test/contract.dll
...

# use contract method `ProposeNewContract` if the chain you are connecting to␣
↪requires create new propose when deploying smart contracts
$ aelf-command send 2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8␣
↪ProposeNewContract
✓ Fetching contract successfully!

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
? Enter the required param <category>: 0
? Enter the required param <code>: /Users/test/contract.dll
...
```

- You must input contract method parameters in the prompting way, note that you can input a relative or absolute path of contract file to pass a file to `aelf-command`, `aelf-command` will read the file content and encode it as a base64 string.
- After call ProposeNewContract, you can get `proposal id` and `proposedContractInputHash` later by running

```
$ aelf-command event␣
↪34184cbc27c95bbc0a1bd676192c3afc380740ab61626e5d428ae17faf9ea984
[Info]:
The results returned by
Transaction: 34184cbc27c95bbc0a1bd676192c3afc380740ab61626e5d428ae17faf9ea984 is:
[
...
  {
    "Address": "pykr77ft9UUKJZLVq15wCH8PinBSjVRQ12sD1Ayq92mKFsJ1i",
    "Name": "ContractProposed",
    "Indexed": [],
    "NonIndexed": "CiIKIK0dKXkwu/HDpZUf/tzjJSfcZ5XznUrE/C0XMtp4liqo",
    "Result": {
      "proposedContractInputHash":
↪"ad1d297930bbf1c3a5951ffedce32527dc6795f39d4ac4fc2d1732da78962aa8"
    }
  },
  {
    "Address": "2JT8xzjR5zJ8xnBvdgBZdSjfbokFSbF5hDdpUCbXeWaJfPDmsK",
    "Name": "ProposalCreated",
    "Indexed": [
      "EiIKIEknWCUo4/KJS/vDAf7u1R6JmLEfAcapRY1BZ9yogawl"
    ],
    "NonIndexed": "CiIKIFb/RK9tR/SjJn0z7d4AjUvw288KCwTRyXSYMryQuC2",
    "Result": {
      "organizationAddress": "ZDcYStbBRACaEQh6K1nqPb2SHKPCTggB9E66onthFoGrVnkfi",
      "proposalId":
↪"56ff44af6d47f4a3267d33edde008d4bf0dbcf0a0b04d1c9749830caf242e0b6"
    }
  }
]
```

- Wait for the organization members to approve your proposal and you can release your proposal by calling `ReleaseApprovedContract`

```
$ aelf-command send 2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8
✔ Fetching contract successfully!
? Pick up a contract method: ReleaseApprovedContract

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
? Enter the required param <proposalId>: proposalId
? Enter the required param <proposedContractInputHash>: proposedContractInputHash
The params you entered is:
{
  "proposalId": proposalNewContract proposalId,
  "proposedContractInputHash": proposedContractInputHash
}
✔ Succeed!
```

- And then you can get code check `proposal id` from event of `ReleaseApprovedContract` transaction.

```
....
  {
    "Address": "2JT8xzjR5zJ8xnBvdgBZdSjfbokFSbF5hDdpUCbXeWaJfPDmsK",
    "Name": "ProposalCreated",
    "Indexed": [
      "EiIKIEknWCUo4/KJS/vDAf7u1R6JmLEfAcapRY1BZ9yogawl"
    ],
    "NonIndexed": "CiIKIAfOf/a3zIillggQjSl2N0Y3aEh8bRGK5ppBrc14CKSn",
    "Result": {
      "organizationAddress": "ZDcYStbBRACaEQh6K1nqPb2SHKPCTggB9E66onthFoGrVnkfi",
      "proposalId":
↪"07ce7ff6b7cc88a59608108d297637463768487c6d118ae69a41adcd7808a4a7"
    }
  }
```

- Wait for the code check pass, then you can release code check proposal by calling `ReleaseCodeCheck`

```
$ aelf-command send 2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8 -a␣
↪28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK -p 123
✔ Fetching contract successfully!
? Pick up a contract method: ReleaseCodeCheckedContract

If you need to pass file contents as a parameter, you can enter the relative␣
↪or absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
? Enter the required param <proposalId>:␣
↪07ce7ff6b7cc88a59608108d297637463768487c6d118ae69a41adcd7808a4a7
? Enter the required param <proposedContractInputHash>:␣
↪ad1d297930bbf1c3a5951ffedce32527dc6795f39d4ac4fc2d1732da78962aa8

The params you entered is:
 {
   "proposalId":␣
↪07ce7ff6b7cc88a59608108d297637463768487c6d118ae69a41adcd7808a4a7,
```

(continues on next page)

(continued from previous page)

```
  "proposedContractInputHash": proposedContractInputHash
}
✓ Succeed!
```

- Finally, you can get deployed contract address later by from event of `ReleaseCodeCheckedContract` transaction.

```
  ....
 {
   ""Address"": ""pykr77ft9UUKJZLVq15wCH8PinBSjVRQ12sD1Ayq92mKFsJ1i"",
   ""Name"": ""ContractDeployed"",
   ""Indexed"": [
     "CiIKIJTPGZ24g4eHwSVNLit8jgjFJeeYCEEYLDpFiCeCT0Bf",
     "EiIKICAU/M9E2AWln6XZSUFrTWR1tXud95vPX1peinPpF7nC"
   ],
   ""NonIndexed"": ""GiIKIK/s1HKVrx1RU5ei3DVJvgc1muE6h2+xyCROHBTfsRqIIAE="",
   ""Result"": {
     ""author"": ""28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK"",
     ""codeHash"": "
↪"2014fccf44d805a59fa5d949416b4d6475b57b9df79bcf5f5a5e8a73e917b9c2"",
     ""address"": ""2LUmicHyH4RXrMjG4beDwuDsiWJESyLkgkwPdGTR8kahRzq5XS"",
     ""version"": 1
   }
 }
```

22.2.7 event - Deserialize the result return by executing a transaction

Only transaction id is required as the parameter.

```
$ aelf-command event fe1974fde291e44e16c55db666f2c747323cdc584d616de05c88c8bae18ecceb
[Info]:
The results returned by
Transaction: fe1974fde291e44e16c55db666f2c747323cdc584d616de05c88c8bae18ecceb is:
[
  {
    "Address": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
    "Name": "ContractDeployed",
    "Indexed": [
      "CiIKIN2O6lDDGWbgbkomYr6+9+2B0JpHsuses3KfLwzHgSmu",
      "EiIKIDXZGwZLKqm78WpYDXuBlyd6Dv+RMjrgOUEnwamfIA/z"
    ],
    "NonIndexed": "GiIKIN2O6lDDGWbgbkomYr6+9+2B0JpHsuses3KfLwzHgSmu",
    "Result": {
      "author": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
      "codeHash": "35d91b064b2aa9bbf16a580d7b8197277a0eff91323ae0394127c1a99f200ff3",
      "address": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8"
    }
  }
]
✓ Succeed!
```

This command get the `Log` field of a transaction result and deserialize the `Log` field with the correspond protobuf descriptors.

A transaction may be related with several `Contract Method`'s events, so the transaction result can include several Logs.

In each item:

- `Address`: the contract address.
- `Name`: name of event published from related contract method.
- `Indexed`: indexed data of event in type of base64
- `NoIndexed`: no indexed data of event in type of base64.
- `Result`: the decoded result, this is readable and you can use it and get what the fields means inside the `Result` by reading the contract documents or contract related protobuf files. In this example, you can read the protobuf file;

22.2.8 send - Send a transaction

```
$ aelf-command send
✔ Enter the the URI of an AElf node ... http://13.231.179.27:8000
✔ Enter a valid wallet address, if you do not have, create one by aelf-command create␣
↪... D3vSjRYL8MpeRpvUDy85ktXijnBe2tHn8NTACsggUVteQCNGP
✔ Enter the password you typed when creating a wallet ... ********
✔ Enter contract name (System contracts only) or the address of contract ... AElf.
↪ContractNames.Token
✔ Fetching contract successfully!
? Pick up a contract method: Transfer

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
? Enter the required param <to>: C91b1SF5mMbenHZTfdfbJSkJcK7HMjeiuwfQu8qYjGsESanXR
? Enter the required param <symbol>: ELF
? Enter the required param <amount>: 100000000
? Enter the required param <memo>: 'test command'
The params you entered is:
{
  "to": "C91b1SF5mMbenHZTfdfbJSkJcK7HMjeiuwfQu8qYjGsESanXR",
  "symbol": "ELF",
  "amount": 100000000,
  "memo": "'test command'"
}
✔ Succeed!
AElf [Info]:
Result:
{
  "TransactionId": "85d4684cb6e4721a63893240f73f675ac53768679c291abeb54974ff4e063bb5"
}
✔ Succeed!
```

```
aelf-command send AElf.ContractNames.Token Transfer '{"symbol": "ELF", "to":
↪"C91b1SF5mMbenHZTfdfbJSkJcK7HMjeiuwfQu8qYjGsESanXR", "amount": "1000000"}'
```

22.2.9 call - Call a read-only method on a contract

```
$ aelf-command call
✓ Enter the the URI of an AElf node ... http://13.231.179.27:8000
✓ Enter a valid wallet address, if you do not have, create one by aelf-command create␣
↪... D3vSjRYL8MpeRpvUDy85ktXijnBe2tHn8NTACsggUVteQCNGP
✓ Enter the password you typed when creating a wallet ... ********
✓ Enter contract name (System contracts only) or the address of contract ... AElf.
↪ContractNames.Token
✓ Fetching contract successfully!
? Pick up a contract method: GetTokenInfo

If you need to pass file contents as a parameter, you can enter the relative or␣
↪absolute path of the file

Enter the params one by one, type `Enter` to skip optional param:
? Enter the required param <symbol>: ELF
The params you entered is:
{
  "symbol": "ELF"
}
✓ Calling method successfully!
AElf [Info]:
Result:
{
  "symbol": "ELF",
  "tokenName": "Native Token",
  "supply": "99732440917954549",
  "totalSupply": "100000000000000000",
  "decimals": 8,
  "issuer": "FAJcKnSpbViZfAufBFzX4nC8HtuT93rxUS4VCMACUwXWYurC2",
  "isBurnable": true,
  "issueChainId": 9992731,
  "burned": "267559132045477"
}
✓ Succeed!
```

```
aelf-command call AElf.ContractNames.Token GetTokenInfo '{"symbol":"ELF"}'
```

22.2.10 get-chain-status - Get the current status of the block chain

```
$ aelf-command get-chain-status
✓ Succeed
{
  "ChainId": "AELF",
  "Branches": {
    "59937e3c16860dedf0c80955f4995a5604ca43ccf39cd52f936fb4e5a5954445": 4229086
  },
  "NotLinkedBlocks": {},
  "LongestChainHeight": 4229086,
  "LongestChainHash":
↪"59937e3c16860dedf0c80955f4995a5604ca43ccf39cd52f936fb4e5a5954445",
  "GenesisBlockHash":
↪"da5e200259320781a1851081c99984fb853385153991e0f00984a0f5526d121c",
  "GenesisContractAddress": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
  "LastIrreversibleBlockHash":
↪"497c24ff443f5cbd33da24a430f5c6c5e0be2f31651bd89f4ddf2790bcbb1906",
```

(continues on next page)

(continued from previous page)

```
  "LastIrreversibleBlockHeight": 4229063,
  "BestChainHash": "59937e3c16860dedf0c80955f4995a5604ca43ccf39cd52f936fb4e5a5954445",
  "BestChainHeight": 4229086
}
```

22.2.11 get-tx-result - Get a transaction result

```
$ aelf-command get-tx-result
✔ Enter the the URI of an AElf node ... http://13.231.179.27:8000
✔ Enter a valid transaction id in hex format ...␣
↪7b620a49ee9666c0c381fdb33f94bd31e1b5eb0fdffa081463c3954e9f734a02
✔ Succeed!
{ TransactionId:
   '7b620a49ee9666c0c381fdb33f94bd31e1b5eb0fdffa081463c3954e9f734a02',
  Status: 'MINED',
  Logs: null,
  Bloom:

↪'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪',
  BlockNumber: 7900508,
  BlockHash:
   'a317c5ecf4a22a481f88ab08b8214a8e8c24da76115d9ddcef4afc9531d01b4b',
  Transaction:
   { From: 'D3vSjRYL8MpeRpvUDy85ktXijnBe2tHn8NTACsggUVteQCNGP',
     To: 'WnV9Gv3gioSh3Vgaw8SSB96nV8fWUNxuVozCf6Y14e7RXyGaM',
     RefBlockNumber: 7900503,
     RefBlockPrefix: 'Q6WLSQ==',
     MethodName: 'GetTokenInfo',
     Params: '{ "symbol": "ELF" }',
     Signature:

↪'JtSpWbMX13tiJD0klMSJQyPBa0aRNFY4hTh3hltdWqhBpv4IRTbjjZfQj39lbBSCOy68vnLg6rUerEcyCsqwfgE=
↪' },
  ReadableReturnValue:
   '{ "symbol": "ELF", "tokenName": "elf token", "supply": "1000000000", "totalSupply
↪": "1000000000", "decimals": 2, "issuer":
↪"2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8", "isBurnable": true }',
  Error: null }
```

22.2.12 get-blk-height - Get the block height

```
$ aelf-command get-blk-height
✔ Enter the the URI of an AElf node ... http://13.231.179.27:8000
> 7902091
```

22.2.13 get-blk-info - Get the block info by a block height or a block hash

You can pass a block height or a block hash to this sub-command.

```
$ aelf-command get-blk-info
✓ Enter the the URI of an AElf node: http://13.231.179.27:8000
✓ Enter a valid height or block hash: 123
✓ Include transactions whether or not: no / yes
{ BlockHash:
   '6034db3e02e283d3b81a4528442988d28997d3828f87cca1a89457b294517372',
  Header:
   { PreviousBlockHash:
      '9d6bcc588c0bc10942899e7ec4536665c86f23286029ed45287babf22c582f5a',
     MerkleTreeRootOfTransactions:
      '7ceb349715787ececa647ad48576467d294de6dcc44d14e19f60c4a91a7a9536',
     MerkleTreeRootOfWorldState:
      'b529e2775283edc39cd4e3f685616085b18bd5521a87ea7904ad99cd2dc50910',
     Extra:
      '[
↪"CkEEJT3FEw+k9cuqv7uruq1fEwQwEjKtYxbXK86wUGrAOH7BgCVkMendLkQZmpEpMgzcz+JXnaVpWtFt3AJcGmGycxL+DggIE
↪AkWcIrCxvCX/
↪Te3fGHVXFxE8xsnfT1HtMgwIoJro6AUQjOa1pQE4TkqCATA0MjUzZGM1MTMwZmE0ZjVjYmFhYmZiYmFiYmFhZDVmMTMwNDMwMT
↪836zXZTbntbqyjgtoBHAEegwIoJro6AUQzOybpgF6DAigmujoBRCk8MG1AnoLCKGa6OgFEOCvuBF6CwihmujoBRCg/
↪JhzegwIoZro6AUQ9Lml1wF6DAihmujoBRDYyOO7AnoMCKGa6OgFEKy+ip8DkAEOEp8CCoIBMDQ4MWMyOWZmYzVlZjI5NjdlMjV
↪", "" ]',
     Height: 123,
     Time: '2019-07-01T13:39:45.8704899Z',
     ChainId: 'AELF',
     Bloom:

↪'00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
     SignerPubkey:

↪'04253dc5130fa4f5cbaabfbbabbaad5f1304301232ad6316d72bceb0506ac0387ec180256431e9dd2e44199a9129320cd
↪' },
  Body:
   { TransactionsCount: 1,
     Transactions:
      [ 'a365a682caf3b586cbd167b81b167979057246a726c7282530554984ec042625' ] } }
```

```
aelf-command get-blk-info␣
↪ca61c7c8f5fc1bc8af0536bc9b51c61a94f39641a93a748e72802b3678fea4a9 true
```

22.2.14 console - Open an interactive console

```
$ aelf-command console
✓ Enter the the URI of an AElf node ... http://13.231.179.27:8000
✓ Enter a valid wallet address, if you do not have, create one by aelf-command create␣
↪... 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✓ Enter the password you typed when creating a wallet ... ********
✓ Succeed!
Welcome to aelf interactive console. Ctrl + C to terminate the program. Double tap␣
↪Tab to list objects



      NAME        | DESCRIPTION
      AElf        | imported from aelf-sdk
```

(continues on next page)

(continued from previous page)

```
      aelf        | instance of aelf-sdk, connect to
                  | http://13.231.179.27:8000
      _account    | instance of AElf wallet, wallet address
                  | is
                  | 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR...
                  | 5oGin1sys6H
```

22.2.15 dapp-server - Start a socket.io server for supplying services for dApps

If you're developing a dApp and you need an environment to hold wallet info and connect to the AElf chain, you can use this sub-command to start a server for dApp local development.

```
$ aelf-command dapp-server
AElf [Info]: DApp server is listening on port 35443

# or listen on a specified port
$ aelf-command dapp-server --port 40334
AElf [Info]: DApp server is listening on port 40334
```

This server uses Socket.io to listen on local port `35443` and you can use aelf-bridge to connect to this server like this:

```
import AElfBridge from 'aelf-bridge';
const bridgeInstance = new AElfBridge({
  proxyType: 'SOCKET.IO',
  socketUrl: 'http://localhost:35443',
  channelType: 'ENCRYPT'
});
// connect to dapp-server
bridgeInstance.connect().then(console.log).catch(console.error);
```

checkout more information in aelf-bridge and aelf-bridge-demo.

CHAPTER 23

Wallet and Block Explorer

23.1 Explorer

Github

Currently, the explorer provides functions such as viewing blocks, transactions, purchasing resources, voting and node campaigning as well as viewing contracts.

23.2 iOS/Android Wallet

iOS/Android Wallet provides basic asset management and cross-chain trading capabilities. It also provides an open application platform for developers to access the wallet according to the usage document of the wallet SDK.

23.3 Web Wallet

Github

The Web Wallet provides basic transaction related functionality.

23.3.1 Explorer-api

To get more informantion by code

Block

Get Block List

```
URL: api/all/blocks?limit={limit}&page={page}
Method: GET
SuccessResponse:
{
    "total": 5850,
    "blocks": [
        {
            "block_hash":
↪"7e1c2fb6d3cc5e8c2cef7d75de9c1adf0e25e9d17d4f22e543fa20f5f23b20e9",
            "pre_block_hash":
↪"6890fa74156b1a88a3ccef1fef72f4f78ff2755ffcd4fb5434ed7b3c153061f5",
            "chain_id": "AELF",
            "block_height": 5719,
            "tx_count": 1,
            "merkle_root_tx":
↪"47eabbc7a499764d0b25c7216ba75fe39717f9866a0716c8a0d1798e64852d84",
            "merkle_root_state":
↪"d14e78dc3c7811b7c17c8b04ebad9e547c35b3faa8bfcc9189b8c12e9f6a4aae",
            "time": "2019-04-27T02:00:34.691118Z"
        },
        {
            "block_hash":
↪"6890fa74156b1a88a3ccef1fef72f4f78ff2755ffcd4fb5434ed7b3c153061f5",
            "pre_block_hash":
↪"f1098bd6df58acf74d8877529702dffc444cb401fc8236519606aa9165d945ae",
            "chain_id": "AELF",
            "block_height": 5718,
            "tx_count": 1,
            "merkle_root_tx":
↪"b29b416148b4fb79060eb80b49bb6ac25a82da2d7a1c5d341e0bf279a7c57362",
            "merkle_root_state":
↪"4dbef401f6d9ed303cf1b5e609a64b1c06a7fb77620b9d13b0e4649719e2fe55",
            "time": "2019-04-27T02:00:34.691118Z"
        },
        {
            "block_hash":
↪"f1098bd6df58acf74d8877529702dffc444cb401fc8236519606aa9165d945ae",
            "pre_block_hash":
↪"1fbdf3a4fb3c41e9ddf25958715815d9d643dfb39e1aaa94631d197e9b1a94bb",
            "chain_id": "AELF",
            "block_height": 5717,
            "tx_count": 1,
            "merkle_root_tx":
↪"776abba03d66127927edc6437d406f708b64c1653a1cc22af9c490aa4f0c22dc",
            "merkle_root_state":
↪"ccc32ab23d619b2b8e0e9b82a53bb66b3a6d168993188b5d3f7f0ac2cb17206f",
            "time": "2019-04-27T02:00:26.690003Z"
        },
    ]
}
```

Get Block List By Bock Hash

```
URL: api/block/transactions?limit={limit}&page={page}&order={order}&block_hash={block_
↪hash}
Method: GET
SuccessResponse:
{
    "transactions": [
        {
            "tx_id": "209ceb8ee88eeb2c55db7783c48ec0b1adf6badba89fc7ddb86e968601027cbb
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 590,
            "address_from": "csoxW4vTJNT9gdvyWS6W7UqEdkSo9pWyJqBoGSnUHXVnj4ykJ",
            "address_to": "2gaQh4uxg6tzyH1ADLoDxvHA14FMpzEiMqsQ6sDG5iHT8cmjp8",
            "params": "",
            "method": "DeploySmartContract",
            "block_hash":
↪"79584a99b7f5da5959a26ff02cbe174d632eb5ef3c6c8d5192de48b6f5584c8d",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T06:47:00.265604Z"
        },
        {
            "tx_id": "d9398736920a5c87ea7cae46c265efa84ac7be4cf8edd37bea54078abef1b44c
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 590,
            "address_from": "2EyPedNTscFK5EwR8FqTrCeW2LZzuPQ7vr18Y5QWuEUApdCkM6",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "NextRound",
            "block_hash":
↪"79584a99b7f5da5959a26ff02cbe174d632eb5ef3c6c8d5192de48b6f5584c8d",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T06:47:00.265604Z"
        }
    ]
}
```

Transactions

Get Transactions List

```
URL: api/all/transactions?limit={limit}&page={limit}
Method: GET
SuccessResponse:
{
    "total": 1179,
    "transactions": [
        {
            "tx_id": "c65d1206e65aaf2e7e08cc818c372ff2c2947cb6cbec746efe6a5e20b7adefa9
↪",
```

(continues on next page)

(continued from previous page)

```
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1064,
            "address_from": "grSAEQ5vJ7UfCN2s1v4fJJnk98bu4SHa2hpQkQ9HT88rmaZLz",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "NextRound",
            "block_hash":
↪"8c922b20164ad3774b56d19673154f383ed89656cbd56433d1681c8c3a4dcab9",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:36.636701Z"
        },
        {
            "tx_id": "4780a7b2737b6f044894719b9bb4cb09862c0b4a7cae267131a0b5c3e7c12850
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1063,
            "address_from": "QUYYqzTQmugruHYmuJVftwmVjnUM82pXnMTnT5jh55qwZKrMw",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "UpdateValue",
            "block_hash":
↪"381114b86b09886f59956851a1d47d8442b29f44f3785dade3c667ca320e23bb",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:36.636701Z"
        },
        {
            "tx_id": "0230385e3f060059d2a62addac09ad6d01f96d32ec076cfbf44c6a3b70c6e092
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1062,
            "address_from": "zizPhdDpQCZxMChMxn1oZ4ttJGJUo61Aocg5BpTYvzLQGmBjT",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "NextRound",
            "block_hash":
↪"06a3ceb783480f4cf5b8402f6749617093d9ea5f9a053f65e86554aeed6d98f4",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:28.635113Z"
        },
    ]
}
```

Get Transactions List By Address

```
URL: api/address/transactions?contract_address={contract_address}&limit={limit}&page=
↪{page}&address={address}
Method: GET
SuccessResponse:
```

(continues on next page)

(continued from previous page)


```
{
    "total": 1179,
    "transactions": [
        {
            "tx_id": "c65d1206e65aaf2e7e08cc818c372ff2c2947cb6cbec746efe6a5e20b7adefa9
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1064,
            "address_from": "grSAEQ5vJ7UfCN2s1v4fJJnk98bu4SHa2hpQkQ9HT88rmaZLz",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "NextRound",
            "block_hash":
↪"8c922b20164ad3774b56d19673154f383ed89656cbd56433d1681c8c3a4dcab9",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:36.636701Z"
        },
        {
            "tx_id": "4780a7b2737b6f044894719b9bb4cb09862c0b4a7cae267131a0b5c3e7c12850
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1063,
            "address_from": "QUYYqzTQmugruHYmuJVftwmVjnUM82pXnMTnT5jh55qwZKrMw",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "UpdateValue",
            "block_hash":
↪"381114b86b09886f59956851a1d47d8442b29f44f3785dade3c667ca320e23bb",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:36.636701Z"
        },
        {
            "tx_id": "0230385e3f060059d2a62addac09ad6d01f96d32ec076cfbf44c6a3b70c6e092
↪",
            "params_to": "",
            "chain_id": "AELF",
            "block_height": 1062,
            "address_from": "zizPhdDpQCZxMChMxn1oZ4ttJGJUo61Aocg5BpTYvzLQGmBjT",
            "address_to": "xw6U3FRE5H8rU3z8vAgF9ivnWSkxULK5cibdZzMC9UWf7rPJf",
            "params": "",
            "method": "NextRound",
            "block_hash":
↪"06a3ceb783480f4cf5b8402f6749617093d9ea5f9a053f65e86554aeed6d98f4",
            "quantity": 0,
            "tx_status": "Mined",
            "time": "2019-04-26T07:18:28.635113Z"
        },
    ]
}
```


TPS

Get TPS Record

```
URL: api/tps/list?start_time={unix_timestamp}&end_time={unix_timestamp}&order={order}
Method: GET
SuccessResponse:
{
    "total": 178,
    "tps": [
        {
            "id": 12498,
            "start": "2019-11-22T01:12:14Z",
            "end": "2019-11-22T01:13:14Z",
            "txs": 1878,
            "blocks": 120,
            "tps": 31,
            "tpm": 1878,
            "type": 1
        },
        {
            "id": 12499,
            "start": "2019-11-22T01:13:14Z",
            "end": "2019-11-22T01:14:14Z",
            "txs": 1889,
            "blocks": 117,
            "tps": 31,
            "tpm": 1889,
            "type": 1
        },
        {
            "id": 12500,
            "start": "2019-11-22T01:14:14Z",
            "end": "2019-11-22T01:15:14Z",
            "txs": 1819,
            "blocks": 114,
            "tps": 30,
            "tpm": 1819,
            "type": 1
        },
        {
            "id": 12501,
            "start": "2019-11-22T01:15:14Z",
            "end": "2019-11-22T01:16:14Z",
            "txs": 1779,
            "blocks": 105,
            "tps": 30,
            "tpm": 1779,
            "type": 1
        }
    ]
}
```

CHAPTER 24

aelf-web-extension

You can get more information in Github

24.1 For User

release version, please waiting

dev version

If you are using qq browser,etc, you can add the extention too.

24.1.1 Notice

Note: Using File:/// protocol may can not use the extenstion // https://developer.chrome.com/extensions/match_patterns Note: Access to file URLs isn't automatic. The user must visit the extensions management page and opt in to file access for each extension that requests it.

24.2 For Dapp Developers

24.2.1 Interaction Flow

- Make sure the user get the Extension
- Connect Chain
- Initialize Contract
- Call contract methods

24.2.2 How to use

If you need complete data structure. you can *click here*

- *Check Extension Demo*
- *GET_CHAIN_STATUS*
- *CALL_AELF_CHAIN*
- *LOGIN*
- *INIT_AELF_CONTRACT*
- *CALL_AELF_CONTRACT / CALL_AELF_CONTRACT_READONLY*
- *CHECK_PERMISSION*
- *SET_CONTRACT_PERMISSION*
- *REMOVE_CONTRACT_PERMISSION*
- *REMOVE_METHODS_WHITELIST*

24.3 Data Format

```
NightElf = {
    histories: [],
    keychain: {
        keypairs: [
            {
                name: 'your keypairs name',
                address: 'your keypairs address',
                mnemonic: 'your keypairs mnemonic',
                privateKey: 'your keupairs privateKey',
                publicKey: {
                    x: 'you keupairs publicKey',
                    y: 'you keupairs publicKey'
                }
            }
        ],
        permissions: [
            {
                chainId: 'AELF',
                contractAddress: 'contract address',
                contractName: 'contract name',
                description: 'contract description',
                github: 'contract github',
                whitelist: {
                    Approve: {
                        parameter1: 'a',
                        parameter2: 'b',
                        parameter3: 'c'
                    }
                }
            }
        ]
    }
}
```

24.3.1 Demo of Checking the Extension

```
let nightElfInstance = null;
class NightElfCheck {
    constructor() {
        const readyMessage = 'NightElf is ready';
        let resovleTemp = null;
        this.check = new Promise((resolve, reject) => {
            if (window.NightElf) {
                resolve(readyMessage);
            }
            setTimeout(() => {
                reject({
                    error: 200001,
                    message: 'timeout / can not find NightElf / please install the␣
↪extension'
                });
            }, 1000);
            resovleTemp = resolve;
        });
        document.addEventListener('NightElf', result => {
            console.log('test.js check the status of extension named nightElf: ',␣
↪result);
            resovleTemp(readyMessage);
        });
    }
    static getInstance() {
        if (!nightElfInstance) {
            nightElfInstance = new NightElfCheck();
            return nightElfInstance;
        }
        return nightElfInstance;
    }
}
const nightElfCheck = NightElfCheck.getInstance();
nightElfCheck.check.then(message => {
    // connectChain -> Login -> initContract -> call contract methods
});
```

24.3.2 GET_CHAIN_STATUS

You can see the demo ./devDemos/test.html. [demo.js just a draft]

If you want to check Token Transfer Demo. You can click here

The methods calls act the same as the methods call of the aelf-sdk.js

Note: '...' stands for omitted data.

```
const aelf = new window.NightElf.AElf({
    httpProvider: [
        'http://192.168.197.56:8101/chain',
        null,
        null,
        null,
        [{
            name: 'Accept',
```

(continues on next page)

(continued from previous page)

```
            value: 'text/plain;v=1.0'
        }]
    ],
    appName: 'Test'
});

aelf.chain.getChainStatus((error, result) => {
    console.log('>>>>>>>>>>>>> connectChain >>>>>>>>>>>>>');
    console.log(error, result);
});

// result = {
//     ChainId: "AELF"
//     GenesisContractAddress: "61W3AF3Voud7cLY2mejzRuZ4WEN8mrDMioA9kZv3H8taKxF"
// }
```

24.3.3 CALL_AELF_CHAIN

```
const txid = 'c45edfcca86f4f528cd8e30634fa4ac53801aae05365cfefc3bfe9b652fe5768';
aelf.chain.getTxResult(txid, (err, result) => {
    console.log('>>>>>>>>>>>>> getTxResult >>>>>>>>>>>>>');
    console.log(err, result);
});

// result = {
//     Status: "NotExisted"
//     TransactionId:
↪"ff5bcd126f9b7f22bbfd0816324390776f10ccb3fe0690efc84c5fcf6bdd3fc6"
// }
```

24.3.4 LOGIN

```
aelf.login({
    appName: 'hzzTest',
    chainId: 'AELF',
    payload: {
        method: 'LOGIN',
        contracts: [{
            chainId: 'AELF',
            contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
            contractName: 'token',
            description: 'token contract',
            github: ''
        }, {
            chainId: 'AELF TEST',
            contractAddress: '2Xg2HKh8vusnFMQsHCXW1q3vys5JxG5ZnjiGwNDLrrpb9Mb',
            contractName: 'TEST contractName',
            description: 'contract description',
            github: ''
        }]
    }
}, (error, result) => {
```

(continues on next page)

(continued from previous page)

```
    console.log('login>>>>>>>>>>>>>>>>>>', result);
});

// keychain = {
//     keypairs: [{
//         name: 'your keypairs name',
//         address: 'your keypairs address',
//         mnemonic: 'your keypairs mnemonic',
//         privateKey: 'your keypairs privateKey'
//         publicKey: {
//             x: 'f79c25eb......',
//             y: '7fa959ed......'
//         }
//     }],
//     permissions: [{
//         appName: 'hzzTest',
//         address: 'your keyparis address',
//         contracts: [{
//             chainId: 'AELF',
//             contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
//             contractName: 'token',
//             description: 'token contract',
//             github: ''
//         }],
//         domain: 'Dapp domain'
//     }]
// }
```

24.3.5 INIT_AELF_CONTRACT

```
// In aelf-sdk.js wallet is the realy wallet.
// But in extension sdk, we just need the address of the wallet.
const tokenContract;
const wallet = {
    address: '2JqnxvDiMNzbSgme2oxpqUFpUYfMjTpNBGCLP2CsWjpbHdu'
};
// It is different from the wallet created by Aelf.wallet.getWalletByPrivateKey();
// There is only one value named address;
aelf.chain.contractAtAsync(
    '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
    wallet,
    (error, result) => {
        console.log('>>>>>>>>>>>>> contractAtAsync >>>>>>>>>>>>>');
        console.log(error, result);
        tokenContract = result;
    }
);

// result = {
//     Approve: ƒ (),
//     Burn: ƒ (),
//     ChargeTransactionFees: ƒ (),
//     ClaimTransactionFees: ƒ (),
//     ....
// }
```

24.3.6 CALL_AELF_CONTRACT / CALL_AELF_CONTRACT_READONLY

```
// tokenContract from the contractAsync
tokenContract.GetBalance.call(
    {
        symbol: 'AELF',
        owner: '65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9'
    },
    (err, result) => {
        console.log('>>>>>>>>>>>>>>>>>>>', result);
    }
);

tokenContract.Approve(
    {
        symbol: 'AELF',
        spender: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
        amount: '100'
    },
    (err, result) => {
        console.log('>>>>>>>>>>>>>>>>>>>', result);
    }
);

// If you use tokenContract.GetBalance.call  this method is only applicable to␣
↪queries that do not require extended authorization validation.(CALL_AELF_CONTRACT_
↪READONLY)
// If you use tokenContract.Approve this requires extended authorization validation␣
↪(CALL_AELF_CONTRACT)

// tokenContract.GetBalance.call(payload, (error, result) => {})
// result = {
//     symbol: "AELF",
//     owner: "65dDNxzcd35jESiidFXN5JV8Z7pCwaFnepuYQToNefSgqk9",
//     balance: 0
// }
```

24.3.7 CHECK_PERMISSION

```
aelf.checkPermission({
    appName: 'hzzTest',
    type: 'address', // if you did not set type, it aways get by domain.
    address: '4WBgSL2fSem9ABD4LLZBpwP8eEymVSS1AyTBCqXjt5cfxXK'
}, (error, result) => {
    console.log('checkPermission>>>>>>>>>>>>>>>>>', result);
});

// result = {
//     ...,
//     permissions:[
//         {
//             address: '...',
//             appName: 'hzzTest',
//             contracts: [{
//                 chainId: 'AELF',
```

(continues on next page)

(continued from previous page)

```
//                  contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
//                  contractName: 'token',
//                  description: 'token contract',
//                  github: ''
//              },
//              {
//                  chainId: 'AELF TEST',
//                  contractAddress: 'TEST contractAddress',
//                  contractName: 'TEST contractName',
//                  description: 'contract description',
//                  github: ''
//              }],
//              domian: 'Dapp domain'
//          }
//      ]
//  }
```

24.3.8 SET_CONTRACT_PERMISSION

```
aelf.setContractPermission({
    appName: 'hzzTest',
    hainId: 'AELF',
    payload: {
        address: '2JqnxvDiMNzbSgme2oxpqUFpUYfMjTpNBGCLP2CsWjpbHdu',
        contracts: [{
            chainId: 'AELF',
            contractAddress: 'TEST contractAddress',
            contractName: 'AAAA',
            description: 'contract description',
            github: ''
        }]
    }
}, (error, result) => {
    console.log('>>>>>>>>>>>>>', result);
});

// keychain = {
//     keypairs: {...},
//     permissions: [{
//         appName: 'hzzTest',
//         address: 'your keyparis address',
//         contracts: [{
//             chainId: 'AELF',
//             contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
//             contractName: 'token',
//             description: 'token contract',
//             github: '',
//             whitelist: {}
//         },
//         {
//             chainId: 'AELF',
//             contractAddress: 'TEST contractAddress',
//             contractName: 'AAAA',
//             description: 'contract description',
//             github: ''
```

(continues on next page)

(continued from previous page)

```
//          }],
//          domain: 'Dapp domain'
//      }]
// }
```

24.3.9 REMOVE_CONTRACT_PERMISSION

```
aelf.removeContractPermission({
    appName: 'hzzTest',
    chainId: 'AELF',
    payload: {
        contractAddress: '2Xg2HKh8vusnFMQsHCXW1q3vys5JxG5ZnjiGwNDLrrpb9Mb'
    }
}, (error, result) => {
    console.log('removeContractPermission>>>>>>>>>>>>>>>>>>>', result);
});

// keychain = {
//     keypairs: {...},
//     permissions: [{
//         appName: 'hzzTest',
//         address: 'your keyparis address',
//         contracts: [{
//             chainId: 'AELF',
//             contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
//             contractName: 'token',
//             description: 'token contract',
//             github: ''
//         }],
//         domain: 'Dapp domain'
//     }]
// }
```

24.3.10 REMOVE_METHODS_WHITELIST

```
aelf.removeMethodsWhitelist({
    appName: 'hzzTest',
    chainId: 'AELF',
    payload: {
        contractAddress: '2Xg2HKh8vusnFMQsHCXW1q3vys5JxG5ZnjiGwNDLrrpb9Mb',
        whitelist: ['Approve']
    }
}, (error, result) => {
    console.log('removeWhitelist>>>>>>>>>>>>>>>>>', result);
});
// keychain = {
//     keypairs: {...},
//     permissions: [{
//         appName: 'hzzTest',
//         address: 'your keyparis address',
//         contracts: [{
//             chainId: 'AELF',
```

(continues on next page)

(continued from previous page)

```
//              contractAddress: '4rkKQpsRFt1nU6weAHuJ6CfQDqo6dxruU3K3wNUFr6ZwZYc',
//              contractName: 'token',
//              description: 'token contract',
//              github: '',
//              whitelist: {}
//          }],
//          domain: 'Dapp domain'
//      }]
// }
```

24.4 For Extension Developers

1. Download the code

```
git clone https://github.com/hzz780/aelf-web-extension.git
```

2. Install dependent

```
npm install
```

3. Run webpack

```
webpack -w
```

4. Add to the browser

```
open development mode, add the webpack output app/public.
```

24.5 Project Information

We use ECDH to use public key to encryt data and private key to decrypt data.

CHAPTER 25

DevOps

25.1 Open source development

We want to stay as open as possible during AElf's development. For this we follow a certain amount rules and guidelines to try and keep the project as accessible as possible. Our project is open source and we publish our code as well as current issues online. It is our responsibility to make it as transparent as possible.

AElf is a collaborative project and welcomes outside opinion and requests/discussion for modifications of the code, but since we work in an open environment all collaborator need to respect a certain standard. We clarify this in the following standard:

-

We encourage collaborators that want to participate to first read the white paper and the documentations to understand the ideas surrounding AElf. Also a look at our code and architecture and the way current functionality has been implemented. After this if any questions remain, you can open an issues on GitHub stating as clearly as possible what you need to clarify.

Finally, any collaborator wanting to participate in the development should open a pull request following our rules. It will be formally reviewed and discussed through GitHub and if validated by core members of AElf, can be merged.

25.2 Deployment

For versioning we use the semver versioning system: https://semver.org

Daily build

Integrated with github we have cron job that will publish the latest version of devs myget packets.

```
- MyGet: https://www.myget.org/gallery/aelf-project-dev
```

Release branch

```
- Nuget: https://www.nuget.org/profiles/AElf
```

25.3 Testing

Testing is one of the most important aspects of software development. Non tested software is difficult to improve. There are two main types of testing that we perform: unit testing and performance testing. The unit testing covers functionality and protocol, which is an essential part of a blockchain system. The performance tests are also very important to show that modifications have not impacted the speed at which our nodes process incoming transactions and blocks.

25.3.1 Unit testing

To ensure the quality of our system and avoid regression, as well as permit safe modifications, we try to cover as much of our functionality as possible through unit tests. We mostly use the xUnit framework and follow generally accepted best practices when testing. Our workflow stipulates that for any new functionality, we cover it with tests and make sure other unit tests.

25.3.2 Perf testing

The performance testing is crucial to AElf since a strong point of our system is speed.

25.4 Monitoring

- Server monitoring: Zabbix monitors instances of aelf metrics like cpu, db…
- Chain monitoring: project on github with Grafana dashboard from Influxdb
- Akka monitoring: monitor actors.

CHAPTER 26

QuickStart

26.1 Manual build & run the sources

This method is not as straightforward as the docker quickstart but is a lot more flexible. If your aim is to develop some dApps it's better you follow these more advanced ways of launching a node. This section will walk you through configuring, running and interacting with an AElf node.

First, if you haven't already done it, clone our repository

```
git clone https://github.com/AElfProject/AElf.git aelf
cd aelf/src
```

Navigate into the newly created **aelf** directory.

26.1.1 Generating the nodes account

First you need to install the **aelf-command** command packet. Open a terminal and enter the following command:

```
npm i -g aelf-command
```

Windows Note: it's possible that you get some errors about python not being installed, you can safely ignore these.

After installing **aelf-command** you can use the following command to create an account/key-pair:

```
aelf-command create
```

The command prompts for a password, enter it and don't forget it. The output of the command should look something like this:

```
Your wallet info is :
Mnemonic            : great mushroom loan crisp ... door juice embrace
Private Key         : e038eea7e151eb451ba2901f7...b08ba5b76d8f288
Public Key          : 0478903d96aa2c8c0...
↪6a3e7d810cacd136117ea7b13d2c9337e1ec88288111955b76ea
```

(continues on next page)

(continued from previous page)

```
Address              : 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H
✓ Save account info into a file? ... no / yes
✓ Enter a password ... ********
✓ Confirm password ... ********
✓
Account info has been saved to "/Users/xxx/.local/share/**aelf**/keys/
↪2Ue31YTuB5Szy7cnr...Gi5uMQBYarYUR5oGin1sys6H.json"
```

In the next steps of the tutorial you will need the **Public Key** and the **Address** for the account you just created. You'll notice the last line of the commands output will show you the path to the newly created key. The **aelf** is the data directory (datadir) and this is where the node will read the keys from.

Note that a more detailed section about the cli can be found *command line interface*.

26.1.2 Node configuration

We have one last step before we can run the node, we have to set up some configuration. Navigate into the **AElf.Launcher** directory:

```
cd AElf.Launcher/
```

This folder contains the default **appsettings.json** file, with some default values already given. There's still some fields that are empty and that need configuring. This will require the information printed during the creation of the account. Open the **appsettings.json** file and edit the following sections.

The account/key-pair associated with the node we are going to run:

```
{
  "Account":
  {
      "NodeAccount": "2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H",
      "NodeAccountPassword": "********"
  },
}
```

The *NodeAccount* field corresponds to the address, you also have to enter the password that you entered earlier.

```
{
  "InitialMinerList" : [
      "0478903d96aa2c8c0...6a3e7d810cacd136117ea7b13d2c9337e1ec88288111955b76ea"
  ],
}
```

This is a configuration that is used to specify the initial miners for the DPoS consensus, for now just configure one, it's the accounts public key that was printed during the account creation.

Note that if your Redis server is on another host listening on a different port than the default, you will also have to configure the connection strings (port/db number):

```
{
  "ConnectionStrings": {
    "BlockchainDb": "redis://localhost:6379?db=1",
    "StateDb": "redis://localhost:6379?db=1"
  },
}
```

We've created an account/key-pair and modified the configuration to use this account for the node and mining, we're now ready to launch the node.

26.1.3 Launch and test

Now we will build and run the node with the following commands:

```
dotnet build AElf.Launcher.csproj --configuration Release
dotnet bin/Release/net6.0/AElf.Launcher.dll > aelf-logs.logs &
cd ..
```

You now should have a node that's running, to check this run the following command that will query the node for its current block height:

```
aelf-command get-blk-height -e http://127.0.0.1:8000
```

26.1.4 Cleanup

To stop the node you can simply find and kill the process:

On mac/Linux:

```
ps -f | grep  [A]Elf.Launcher.dll | awk '{print $2}'
```

On Windows (Powershell):

```
Get-CimInstance Win32_Process -Filter "name = 'dotnet.exe'" | select CommandLine,
↪ProcessId | Where-Ob
ject {$_.CommandLine -like "*AElf.Launcher.dll"} | Stop-Process -ID {$_.ProcessId}
```

If needed you should also clean your redis database, with either of the following commands:

```
redis-cli FLUSHALL (clears all dbs)
```

```
redis-cli -n <database_number> FLUSHDB (clear a specified db)
```

26.1.5 Extra

For reference and after you've started a node, you can get infos about an account with the *aelf-command console* command:

```
aelf-command console -a 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR5oGin1sys6H

✓ Enter the password you typed when creating a wallet ... ********
✓ Succeed!
Welcome to aelf interactive console. Ctrl + C to terminate the program. Double tap␣
↪Tab to list objects



      NAME       | DESCRIPTION
      AElf       | imported from aelf-sdk
      aelf       | the instance of an aelf-sdk, connect to
```

(continues on next page)

(continued from previous page)

```
              | http://127.0.0.1:8000
    _account   | the instance of an AElf wallet, address
              | is
              | 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR...
              | 5oGin1sys6H
```

CHAPTER 27

Developing smart contracts

AElf is part of a relatively new software type called the blockchain. From a high-level perspective, a blockchain is a network of interconnected nodes that process transactions in order to form blocks. Transactions are usually broadcast to the network by sending them to a node; this node verifies the transaction, and if it's correct will broadcast it to other nodes. The client that sent the transaction can be of many types, including a browser, script or any client that can connect and send HTTP requests to a node.

Internally blockchains keep a record of all the transactions ever executed by the network, and these transactions are contained in cryptographically linked blocks. AElf uses a DPoS consensus type in which miners collect transactions and, according to a schedule, package them into blocks that are broadcast to the network. These linked blocks effectively constitute the blockchain (here, blockchain refers to the data structure rather than the software). In AElf the transaction and blocks are usually referred to as **chain data**.

Smart contracts are pieces of code that can be executed by transactions, and that will usually modify their associated state. In other words, the execution of transactions modifies the current values of the contracts state. The set of all the state variables of all the contracts is referred to as a **state data**.

27.1 Contracts in AElf

Conceptually, AElf smart contracts are entities composed of essentially three things: **action** methods, **view** methods, and the contracts **state**. Actions represent logic that modifies the state of the contract, and views are used to fetch the current state of the contract without modifying it. Theses two types of methods are executed when a transaction is being processed by a node, usually when executing a block or producing it.

In practice, an aelf contract is written in C# with some parts that are generated from a **protobuf definition**. The protobuf is used to define the contract's methods and data types. By using a custom plugin, the protobuf compiler generates the C# code that is later extended by the contract author to implement logic.

27.2 Development

Currently, the primary language supported by an AElf node is C#. The provided **C# SDK** contains all essential elements for writing smart contracts, including communication with the execution context, access to state and storage primitives.

Writing a contract boils down to creating a protobuf definition and a C# project (referred to sometimes as a Class Library in the C# world) and referencing the SDK. Only a small subset of the C# language is needed to develop a contract.

This series of articles mainly uses AElf Boilerplate as a smart contract development framework. It takes care of the build process for the contract author and provides some well-defined location to place the contract files. The first article will show you how to set up this environment. After the setup, the next three articles will walk you through creating, testing, and deploying a contract. Later articles will focus on exposing more complex functionality.

27.2.1 Setup

AElf Boilerplate is the go-to environment for creating and testing smart contracts. It takes care of including your contract files in the build system and linking the appropriate development SDK. Boilerplate also takes care of generating the csharp code from the proto definition.

This article will get you started with development on Boilerplate. It contains the following items: - how to clone, build, and run AElf Boilerplate. - how to run the Hello World contract tests. - a brief presentation of Boilerplate.

Environment

IDE

Strictly speaking, you don't need an IDE for this tutorial, but it is highly recommended. If you don't already have one you can try Visual Studio Code (vscode) with the C# extension: - installation instructions for vscode here. - working with C# extension here.

You can, of course, use your favorite C# IDE, most of the steps described here and in later articles do not need IDE support.

Clone the repository

The following command will clone AElf Boilerplate into a **aelf-boilerplate** folder with Boilerplate's code inside it, open a terminal and enter the following command:

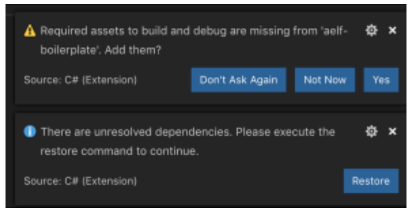
```
git clone https://github.com/AElfProject/aelf-boilerplate
```

The boilerplate repo contains a framework for easy smart contract development as well as examples (some explained in this series of articles).

Build and run

Open the project

If not already done, open vscode and open the **aelf-boilerplate** folder. If asked to add some "required assets" say **yes**. There may also be some dependencies to restore: for all of them, choose **Restore**.



Open vscode's **Integrated Terminal** and build the project with the following command. Note: you can find out more about vscode's terminal here.

Install script

As stated earlier, Boilerplate takes care of the C# code generation and thus has a dependency on protobuf. If you don't already have it installed, run the following script from withing the **aelf-boilerplate** folder:

```
# Mac or Linux
sh chain/scripts/install.sh

# Windows
# open a PowerShell console as administrator
chain/scripts/install.ps1
```

{% hint style="info" %} If you prefer or have problems, you can refer to the following guide to manually install protobuf on your system. {% endhint %}

Build and run

The next step is to build Boilerplate and all the contracts to ensure everything is working correctly. Once everything is built, we'll run Boilerplate's internal node.

```
# enter the Launcher folder and build
cd chain/src/AElf.Boilerplate.Launcher/

# build
dotnet build

# run the node
dotnet run --no-build bin/Debug/net6.0/AElf.Boilerplate.Launcher
```

{% hint style="warning" %} When running Boilerplate, you might see some errors related to an incorrect password, to solve this, you need to backup your `data-dir/keys/` folder and start with an empty keys folder. Once you've cleaned the keys, stop and restart the node with the `dotnet run` command shown above. {% endhint %}

At this point, the smart contracts have been deployed and are ready to be called (Boilerplate has a functioning API). You should see the node's logs in the terminal and see the node producing blocks. You can now stop the node by killing the process (usually **control-c** or **ctrl-c** in the terminal).

Run tests

Boilerplate makes it easy to write unit tests for your contracts. Here we'll take the tests of the Hello World contract included in Boilerplate as an example. To run the tests, navigate to the **AElf.Contracts.HelloWorldContract.Test** folder and run:

```
cd ../../test/AElf.Contracts.HelloWorldContract.Test/
dotnet test
```

The output should look somewhat like this, meaning that the tests have successfully executed:

```
Test Run Successful.
Total tests: 1
     Passed: 1
 Total time: 2.8865 Seconds
```

At this point, you have successfully downloaded, built, and run Boilerplate. You have also run the HelloWorld contract's tests that are included in Boilerplate. Later articles will show you how to add a contract and its tests and add it to the deployment process.

More on Boilerplate

Boilerplate is an environment that is used to develop smart contracts and dApps. After writing and testing your contract on Boilerplate, you can deploy it to a running AElf chain. Internally Boilerplate will run an AElf node that will automatically have your contract deployed on it at genesis.

Boilerplate is composed of two root folders: **chain** and **web**. This series of tutorial articles focuses on contract development so we'll only go into the details of the **chain** part of Boilerplate. Here is a brief overview of the folders:

```
.
└── chain
    ├── src
    ├── contract
    │   └── AElf.Contracts.HelloWorldContract
    │       ├── AElf.Contracts.HelloWorldContract.csproj
    │       ├── HelloWorldContract.cs
    │       ├── HelloWorldContractState.cs
    │       └── ...
    ├── protobuf
    │   ├── hello_world_contract.proto
    │   └── ...
    ├── test
    │   └── AElf.Contracts.HelloWorldContract.Test
    │       ├── AElf.Contracts.HelloWorldContract.Test.csproj
    │       └── HelloWorldContractTest.cs
    └── ...
```

The hello world contract and its tests are split between the following folders: - **contract**: this folder contains the csharp projects (.csproj) along with the contract implementation (.cs files). - **protobuf**: contains the .proto definition of the contract. - **test**: contains the test project and files (basic xUnit test project).

You can use this layout as a template for your future smart contracts. Before you do, we recommend you follow through all the articles of this series.

{% hint style="info" %} You will also notice the **src** folder. This folder contains Boilerplate's modules and the executable for the node. {% endhint %}

Next

You've just seen a short introduction on how to run a smart contract that is already included in Boilerplate. The next article will show you a complete smart contract and extra content on how to organize your code and test files.

{% hint style="warning" %} All production contracts (contracts destined to be deployed to a live chain) must go through a complete review process by the contract author and undergo proper testing. It is the author's responsibility to check the validity and security of his contract. The author should not simply copy the contracts contained in Boilerplate; it's the author's responsibility to ensure the security and correctness of his contracts. {% endhint %}

27.2.2 Transaction execution context

This article will present some of the functionality available to smart contract developers that can help them implement common scenarios.

When executing, transactions trigger the logic contained inside smart contracts. The smart contract execution is mostly sandboxed (it's an isolated environment), but some elements are accessible to the smart contract author through the **execution context**.

Before we get started with the examples, it's important to know a little about the execution model of transactions; this will help you understand some concepts explained in this article. As a reminder this is what a transaction in AElf looks like (simplified):

```
message Transaction {
    Address from = 1;            // the address of the signer
    Address to = 2;              // the address of the target contract
    int64 ref_block_number = 3;  // the block number
    bytes ref_block_prefix = 4;  // the block prefix info
    string method_name = 5;      // the method to execute
    bytes params = 6;            // the parameters to pass to the method
    bytes signature = 10000;     // the signature of this transaction (by the Sender)
}
```

When users create and send a transaction to a node, it will eventually be packaged in a block. When this block is executed, the transactions it contains are executed one by one.

Each transaction can generate new transactions called inline transactions (more on this in the next article). When this happens, the generated inline transactions are executed right after the transaction that generated them. For example, let's consider the following scenario: a block with two transactions, let's say **tx1** and **tx2**, where **tx1** performs two inline calls. In this situation, the order of execution will be the following:

‘”

1. execute tx1
2. • Execute first inline
3. • Execute second Inline
4. execute tx2 ‘”

This is important to know because, as we will see next, some of the execution context's values change based on this logic.

Origin, Sender and Self

- **Origin**: the address of the sender (signer) of the transaction being executed. Its type is an AElf address. It corresponds to the **From** field of the transaction. This value never changes, even for nested inline calls. This means that when you access this property in your contract, it's value will be the entity that created the transaction (user or smart contract through an inline call)
- **Self**: the address of the contract currently being executed. This changes for every transaction and inline transaction.

- **Sender**: the address sending the transaction. If the transaction execution does not produce any inline transactions, this will always be the same. But if one contract calls another with an inline transaction, the sender will be the contract that is calling.

To use these values, you can access them through the **Context** property.

```
Context.Origin
Context.Sender
Context.Self
```

Useful properties

There are other properties that can be accessed through the context:

- transaction ID: this is the id of the transaction that is currently being executed. Note that inline transactions have their own ID.
- chain ID: the ID of the current chain, this can be useful in the contract that needs to implement cross-chain scenarios.
- current height: the height of the block that contains the transaction currently executing.
- current block time: the time included in the header of the current block.
- previous block hash: the hash of the block that precedes the current.

Useful methods

Logging and events:

Fire log event - these are logs that can be found in the transaction result after execution.

```
public override Empty Vote(VoteMinerInput input)
{
    // for example the election system contract will fire a 'voted' event
    // when a user calls vote.
    Context.Fire(new Voted
    {
        VoteId = input.VoteId,
        VotingItemId = votingRecord.VotingItemId,
        Voter = votingRecord.Voter
        //...
    });
}
```

Application logging - when writing a contract, it is useful to be able to log some elements in the applications log file to simplify development. Note that these logs are only visible when the node executing the transaction is build in **debug** mode.

```
private Hash AssertValidNewVotingItem(VotingRegisterInput input)
{
    // this is a method in the voting contract that will log to the applications log␣
↪file
    // when a 'voting item' is created.
    Context.LogDebug(() => "Voting item created by {0}: {1}", Context.Sender,␣
↪votingItemId.ToHex());
```

(continues on next page)

(continued from previous page)

```
    // ...
}
```

Get contract address

It's sometimes useful to get the address of a system contract; this can be done as follows:

```
    public override Empty AddBeneficiary(AddBeneficiaryInput input)
    {
        // In the profit contract, when adding a 'beneficiary', the method will get␣
↪the address of the token holder
        // contract from its name, to perform an assert.

        Assert(Context.Sender == Context.
↪GetContractAddressByName(SmartContractConstants.TokenHolderContractSystemName),
        "Only manager can add beneficiary.");
    }
```

Recovering the public key

Recovering the public key: this can be used for recovering the public key of the transaction Sender.

```
public override Empty Vote(VoteMinerInput input)
{
    // for example the election system contract will use the public key of the sender
    // to keep track of votes.
    var recoveredPublicKey = Context.RecoverPublicKey();
}
```

27.2.3 Internal contract interactions

There are essentially two reasons for interacting with other contracts:

1. to query their state.
2. to create an inline transaction, that is, a new transaction which will be executed after the original transaction.

Both of the two operations can be done in two ways:

1. using the **transaction execution context**.
2. adding a **Contract Reference State** to the contract, then using **CSharpSmartContract.State** to call methods.

Using the Context

Query state from other contracts

Let's see how to call the **GetCandidates** method of the **Election Contract** and get the return value directly in your contract code with the **Context** property that is available in every smart contract.

```
using AElf.Sdk.CSharp;
using AElf.Contracts.Election;
...
// your contract code needs the candidates
var electionContractAddress =
    Context.GetContractAddressByName(SmartContractConstants.
↪ElectionContractSystemName);

// call the method
var candidates = Context.Call<PubkeyList>(electionContractAddress, "GetCandidates",␣
↪new Empty());

// use **candidates** to do other stuff...
```

There are several things to know before writing such code:

- Because this code references a type (**PubkeyList**) originally defined in the Election Contract (types are defined in a proto file, in this case, **election_contract.proto**), you at least need to reference messages defined in the .proto file in your contracts project.

Add these lines to your csproj file:

```
    <ItemGroup>
        <ContractMessage Include="..\..\protobuf\election_contract.proto">
            <Link>Protobuf\Proto\reference\election_contract.proto</Link>
        </ContractMessage>
    </ItemGroup>
```

The **ContractMessage** tag means you just want to reference the messages defined in the specified .proto file.

- The `Call` method takes the three following parameters:
  - *address*: the address of the contract you're seeking to interact with.
  - *methodName*: the name of the method you want to call.
  - *message*: the argument for calling that method.
- Since the `Election Contract` is a system contract which deployed at the very beginning of AElf blockchain, we can get its address directly from the `Context` property. If you want to call contracts deployed by users, you may need to obtain the address in another way (like hard code).

To send an inline transaction

Imagine you want to transfer some tokens from the contract you're writing, the necessary step is sending an inline transaction to `MultiToken Contract`, and the `MethodName` of this inline transaction needs to be `Transfer` .

```
var tokenContractAddress = Context.GetContractAddressByName(SmartContractConstants.
↪TokenContractSystemName);
Context.SendInline(tokenContractAddress, "Transfer", new TransferInput
{
    To = toAddress/* The address you wanna transfer to*/,
    Symbol = Context.Variables.NativeSymbol,// You will get "ELF" if this contract is␣
↪deployed in AElf main chain.
    Amount = 100_000_00000000,// 100000 ELF tokens.
    Memo = "Gift."// Optional
});
```

Again, because you have to reference a message defined by the m=Multi-Token contract proto file, you need to add these lines to the csproj file of your contract project.

```
    <ItemGroup>
        <ContractMessage Include="..\..\protobuf\token_contract.proto">
            <Link>Protobuf\Proto\reference\token_contract.proto</Link>
        </ContractMessage>
    </ItemGroup>
```

This inline transaction will be executed after the execution of the original transaction. Check other documentation for more details about the inline transactions.

Using `Contract Reference State`

Using `Contract Reference State` is more convenient than using `Context` to do the interaction with another contract. Follow these three steps of preparation:

1. Add a related proto file(s) of the contract you want to call or send inline transactions to and rebuild the contract project. (like before, but we need to change the MSBUILD tag name, we'll see this later.)
2. Add an internal property of `XXXContractReferenceState` type to the State class of your contract.
3. Set the contract address to the `Value` of property you just added in step 2.

Let's see a demo that implements these steps: check the balance of ELF token of the current contract, if the balance is more significant than 100 000, request a random number from `AEDPoS Contract`.

First, reference proto files related to `MultiToken Contract` and `acs6.proto` (random number generation).

```
<ItemGroup>
    <ContractReference Include="..\..\protobuf\acs6.proto">
        <Link>Protobuf\Proto\reference\acs6.proto</Link>
    </ContractReference>
    <ContractReference Include="..\..\protobuf\token_contract.proto">
        <Link>Protobuf\Proto\reference\token_contract.proto</Link>
    </ContractReference>
</ItemGroup>
```

After rebuilding the contract project, we'll see following files appear in the Protobuf/Generated folder:

- Acs6.c.cs
- Acs6.g.cs
- TokenContract.c.cs
- TokenContract.g.cs

As you may guess, the entities we will use are defined in files above.

Here we will define two `Contract Reference States`, one for the token contract and one for the random number provider.

```
using AElf.Contracts.MultiToken;
using Acs6;

...

// Define these properties in the State file of current contract.
```

(continues on next page)

(continued from previous page)

```
internal TokenContractContainer.TokenContractReferenceState TokenContract { get; set;␣
↪}
internal RandomNumberProviderContractContainer.
↪RandomNumberProviderContractReferenceState ACS6Contract { get; set }
```

Life becomes very easy if we have these `XXXContractReferenceState` instances. Check the implementation.

```
// Set the Contract Reference States address before using it (again here, we already␣
↪have the system addresses for the token and ac6 contracts).
if (State.TokenContract.Value == null)
{
    State.TokenContract.Value =
        Context.GetContractAddressByName(SmartContractConstants.
↪TokenContractSystemName);
}
if (State.ACS6Contract.Value == null)
{
    // This means we use the random number generation service provided by `AEDPoS␣
↪Contract`.
    State.ACS6Contract.Value =
        Context.GetContractAddressByName(SmartContractConstants.
↪ConsensusContractSystemName);
}

// Use `Call` method to query states from multi-token contract.
var balance = State.TokenContract.GetBalance.Call(new GetBalanceInput
{
    Owner = Context.Self,// The address of current contract.
    Symbol = "ELF"// Also, you can use Context.Variables.NativeSymbol if this␣
↪contract will deployed in AElf main chain.
});
if (balance.Balance > 100_000)
{
    // Use `Send` method to generate an inline transaction.
    State.ACS6Contract.RequestRandomNumber.Send(new RequestRandomNumberInput());
}
```

As you can see, it is convenient to call a method by using state property like this: State.**Contract**.**method**.Call(**input**).