AElf

Release release/1.0.0-preview3

Dec 18, 2020

Contents

CHAPTER 1

Welcome to AElf's official documentation.

This GitBook 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 Dependencies

Before you jump in to the getting started 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 the official link with full instructions.

This page is divided into two sections: the first concerns the common dependencies that are needed for running a node. The second shows the extra dependencies needed for building the sources and development tools.

2.1.1 Common dependencies

Pre-setup for Windows users

A convenient tool for Windows users is **Chocolatey** for installing dependencies. Follow the installation instructions below (see here for more details Chocolatey installation):

Open and administrative Powershell and enter the following commands:

```
Set-ExecutionPolicy AllSigned
or
Set-ExecutionPolicy Bypass -Scope Process

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

Later, **Chocolatey** can be very useful for installing dependencies on Windows systems.

Pre-setup for macOS users

It is highly recommended that you install **Homebrew (or simply Brew)** to quickly and easily setup dependencies (see here for more details Homebrew install page). Open a terminal and execute the following command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Node js

Next install nodejs by following the instructions here (see here for more details Nodejs):

On macOS:

```
brew install node
```

On Windows:

```
choco install nodejs
```

On Linux:

```
sudo apt-get install nodejs
```

Database

We currently support two key-value databases to store our nodes data: **Redis** or **SSDB**. Both work well, it's your decision:

Redis (recommended for the tutorials):

Depending on your platform, enter one of the following commands (see here for more details Redis download page):

On Windows:

```
choco install redis-64
```

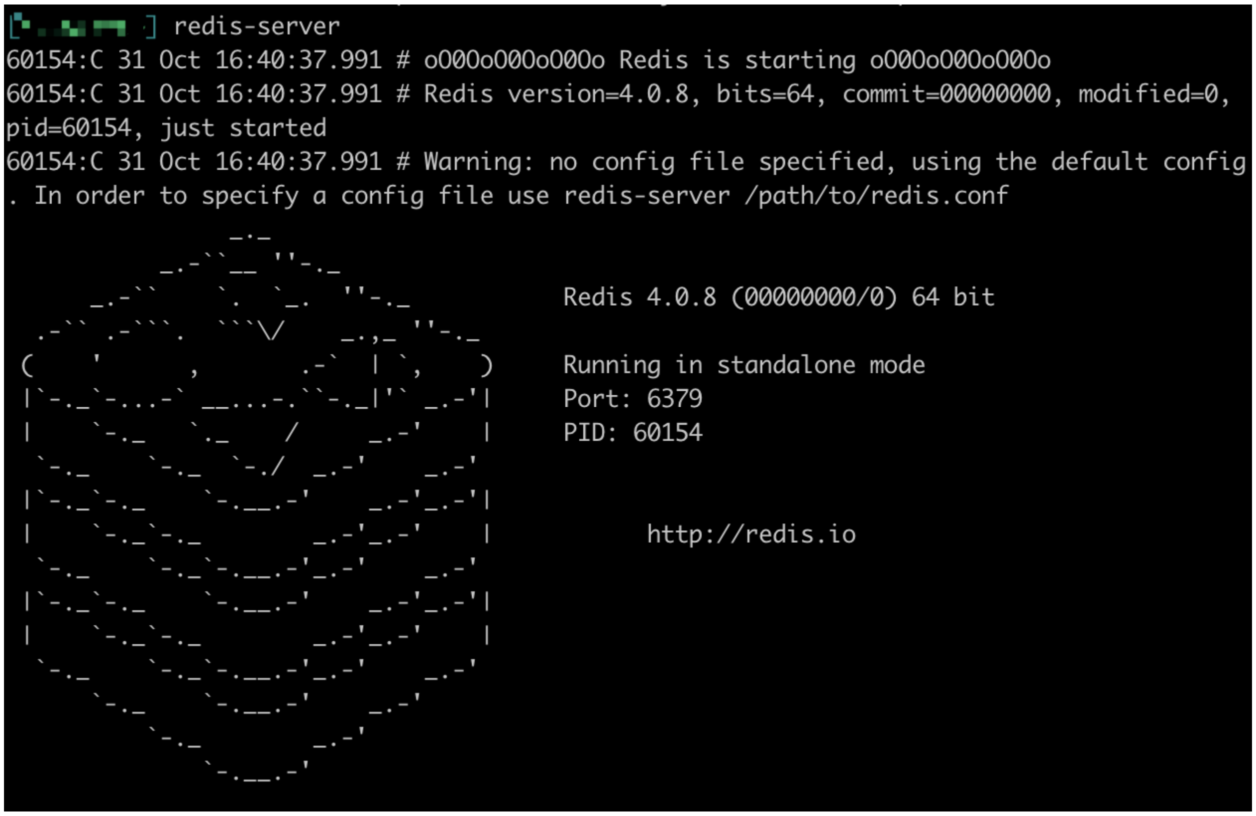
On macOS:

```
brew install redis
```

On Linux:

```
sudo apt install redis-server
```

To test the installation (all platforms) you can just open a terminal and type `redis-server`. This will show you the servers welcome page as well as the port it's listening on:



```
[redacted] redis-server
60154:C 31 Oct 16:40:37.991 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
60154:C 31 Oct 16:40:37.991 # Redis version=4.0.8, bits=64, commit=00000000, modified=0,
pid=60154, just started
60154:C 31 Oct 16:40:37.991 # Warning: no config file specified, using the default config
. In order to specify a config file use redis-server /path/to/redis.conf
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 4.0.8 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in standalone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 60154
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'
```

SSDB

Depending on your platform, enter one of the following commands (see here for more details SSDB):

NOTE: On a Windows machine we highly recommend you use Redis. This is an extract from the official website:

```
Do not run SSDB server on Windows system for a production environment. If you wish to␣
↪stick with Windows system, please run a Linux virtual machine on Windows, and run␣
↪SSDB server on that Linux.
```

On macOS:

```
brew install ssdb
```

On Linux:

```
wget --no-check-certificate https://github.com/ideawu/ssdb/archive/master.zip
unzip master
cd ssdb-master
make
# optional, install ssdb in /usr/local/ssdb
sudo make install
```

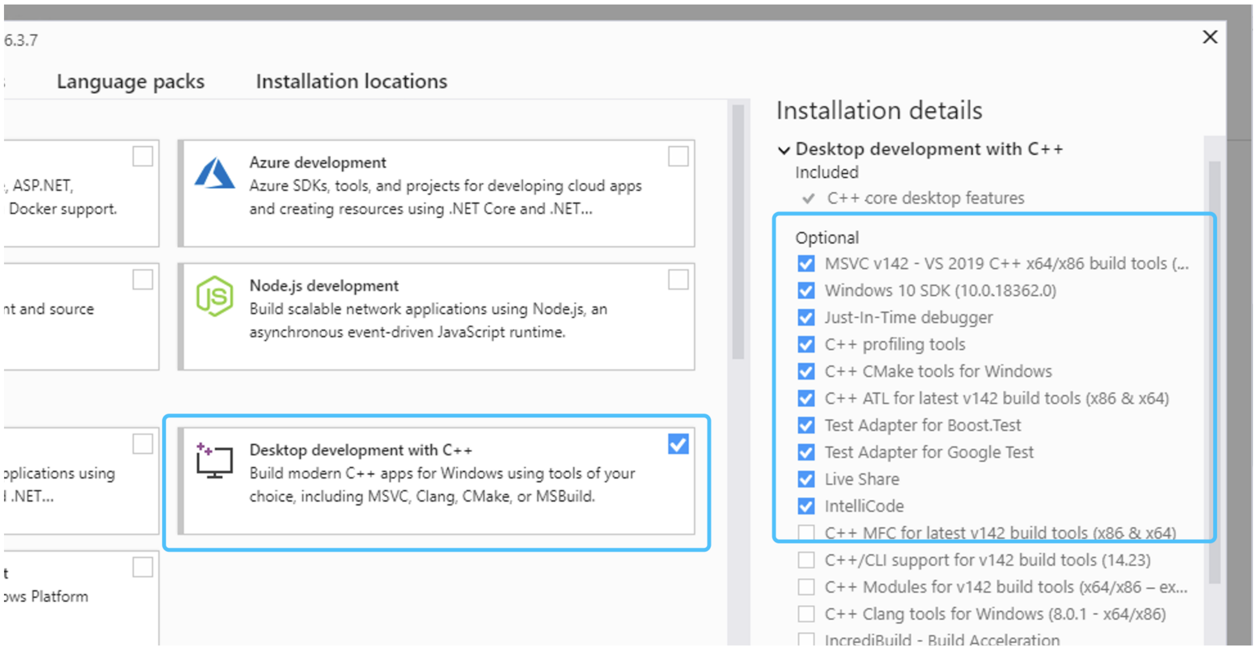
2.1.2 Building sources and development tools

You only need to follow this section if you intend to build aelf from the sources available on Github or if you plan on doing smart contract development.

Windows build tools

A dependency needed to build **AElf** from the command line under Windows is **Visual Studio Build Tools**. The easiest way is to use the **Visual Studio Installer**:

If you already have an edition of **Visual Studio** installed, open the **Visual Studio Installer** and add the **Desktop development with C++** workload:



6.3.7

Language packs Installation locations

, ASP.NET, Docker support.

Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET Core and .NET...

nt and source

Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

pplications using .NET...

Desktop development with C++ ✓
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

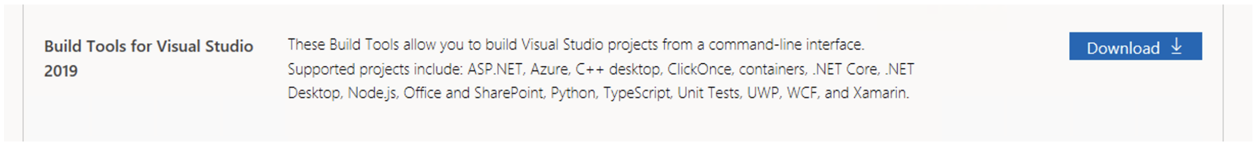
ows Platform

Installation details

Desktop development with C++
Included
✓ C++ core desktop features
Optional
✓ MSVC v142 - VS 2019 C++ x64/x86 build tools (...
✓ Windows 10 SDK (10.0.18362.0)
✓ Just-In-Time debugger
✓ C++ profiling tools
✓ C++ CMake tools for Windows
✓ C++ ATL for latest v142 build tools (x86 & x64)
✓ Test Adapter for Boost.Test
✓ Test Adapter for Google Test
✓ Live Share
✓ IntelliCode
☐ C++ MFC for latest v142 build tools (x86 & x64)
☐ C++/CLI support for v142 build tools (14.23)
☐ C++ Modules for v142 build tools (x64/x86 – ex...
☐ C++ Clang tools for Windows (8.0.1 - x64/x86)
☐ IncrediBuild - Build Acceleration

If you don't have any of the Visual Studio editions installed:

- you can download it here Visual Studio Community Edition for free and after the installation add the **Desktop development with C++** workload.
- or if you don't need or want a full blown installation of **Visual Studio**, you can download the build tools here: Download Page. Scroll down and under the section *Tools for Visual Studio 2019* download the build tools for Visual Studio:

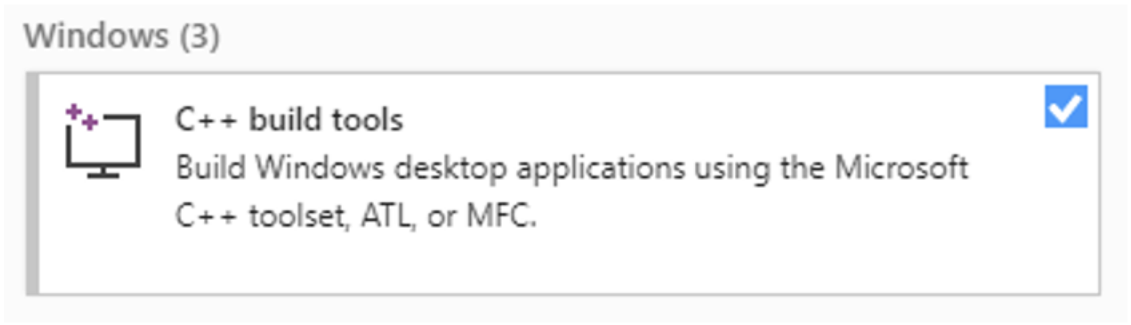


Build Tools for Visual Studio 2019

These Build Tools allow you to build Visual Studio projects from a command-line interface. Supported projects include: ASP.NET, Azure, C++ desktop, ClickOnce, containers, .NET Core, .NET Desktop, Node.js, Office and SharePoint, Python, TypeScript, Unit Tests, UWP, WCF, and Xamarin.

Download

After the installation open **Visual Studio Installer**, locate and install the *C++ build tools*.



Windows (3)

C++ build tools ✓
Build Windows desktop applications using the Microsoft C++ toolset, ATL, or MFC.

Git

If you want to run a node or use our custom smart contract environment, at some point you will have to clone (download the source code) from AElf's repository. For this you will have to use **Git** since we host our code on GitHub.

Click the following link to download Git for your platform (see here for more details Getting Started - Installing Git):

On macOS:

```
brew install git
```

On Windows:

```
choco install git
```

On Linux:

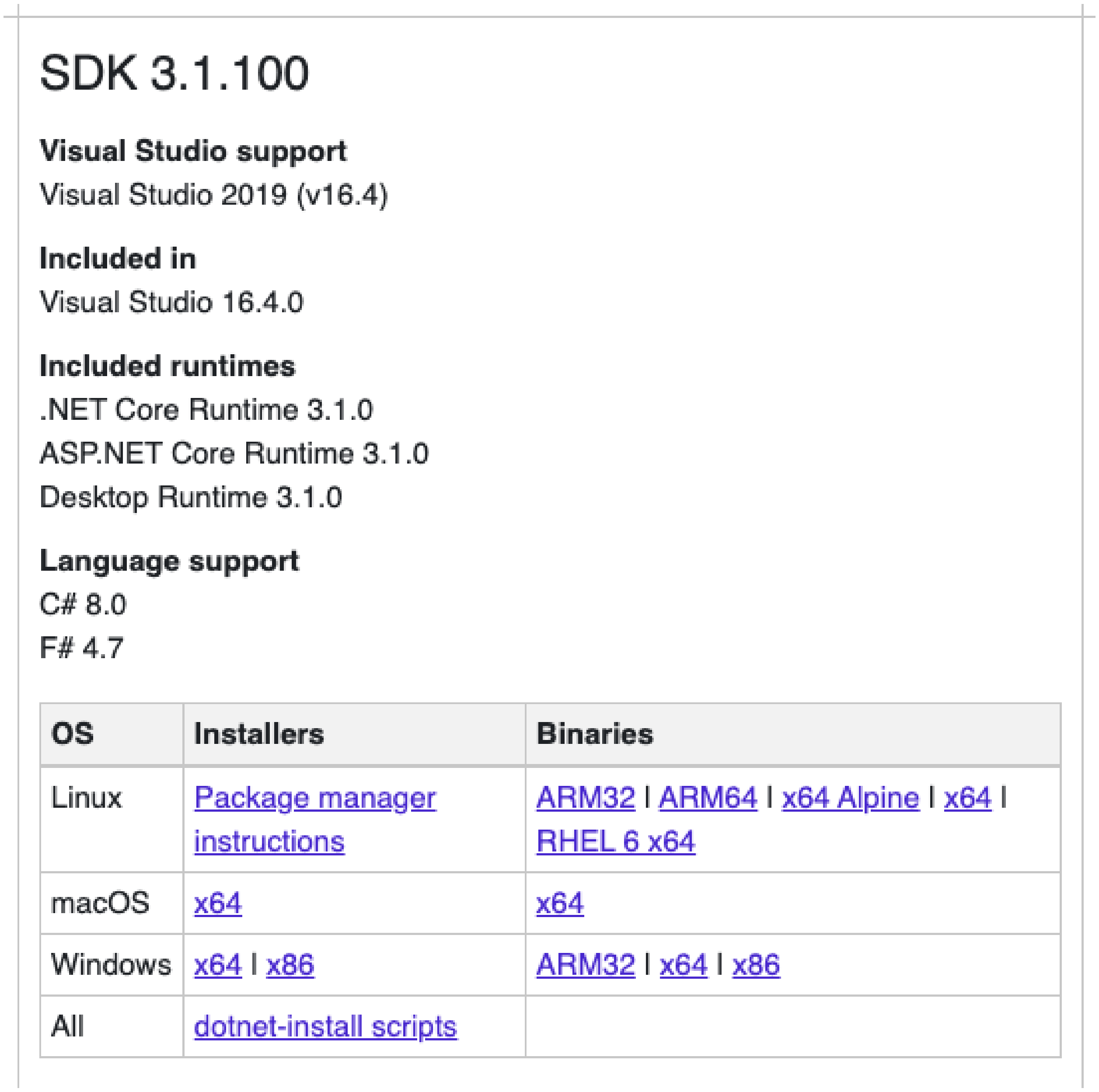
```
sudo apt install git-all
```

Development framework - dotnet core sdk

Most of AElf is developed with dotnet core, so you will need to download and install the .NET Core SDK before you start:

Download .NET Core 3.1

For now AElf depends on version 3.1 of the SDK, on the provided link find the download for your platform (for Windows and macOS the installer for x64 is the most convenient if your platform is compatible - most are these days), the page looks like this:



SDK 3.1.100

Visual Studio support

Visual Studio 2019 (v16.4)

Included in

Visual Studio 16.4.0

Included runtimes

.NET Core Runtime 3.1.0

ASP.NET Core Runtime 3.1.0

Desktop Runtime 3.1.0

Language support

C# 8.0

F# 4.7

OS	Installers	Binaries
Linux	Package manager instructions	ARM32 | ARM64 | x64 Alpine | x64 | RHEL 6 x64
macOS	x64	x64
Windows	x64 | x86	ARM32 | x64 | x86
All	dotnet-install scripts	

Wait for the download to finish, launch the installer and follow the instructions (for AElf all defaults provided in the installer should be correct).

To check the installation, you can open a terminal and run the `dotnet` command. If everything went fine it will show you dotnet options for the command line.

Protobuf

Depending on your platform, enter one of the following commands (see here for more details Protobuf Github):

On Windows, open a **Powershell** and enter the following commands:

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

On Linux:

```
# Make sure you grab 3.11.4 for protoc
curl -OL https://github.com/google/protobuf/releases/download/v3.11.4/protoc-3.11.4-
↪linux-x86_64.zip

# Unzip
unzip protoc-3.11.4-linux-x86_64.zip -d protoc3

# Move protoc to /usr/local/bin/
sudo mv protoc3/bin/* /usr/local/bin/

# Move protoc3/include to /usr/local/include/
sudo mv protoc3/include/* /usr/local/include/

# Optional: change owner
sudo chown ${USER} /usr/local/bin/protoc
sudo chown -R ${USER} /usr/local/include/google
```

on macOS:

```
# Make sure you grab 3.11.4 for protoc
curl -OL https://github.com/protocolbuffers/protobuf/releases/download/v3.11.4/protoc-
↪3.11.4-osx-x86_64.zip

# Unzip
unzip protoc-3.11.4-osx-x86_64.zip -d protoc3

# Move protoc to /usr/local/bin/
sudo mv protoc3/bin/* /usr/local/bin/

# Move protoc3/include to /usr/local/include/
sudo mv protoc3/include/* /usr/local/include

# Optional: change owner
sudo chown ${USER} /usr/local/bin/protoc
sudo chown -R ${USER} /usr/local/include/google
```

2.2 Running a node with Docker

A pre-requisite to this tutorial is to install Docker on your system.

2.2.1 Pull AElf Docker image

After you have completed the Docker installation, you can pull the latest version of the official AElf image with the following command:

```
docker pull aelf/node
```

While downloading you can make sure your Redis database instance is ready.

2.2.2 Create configuration

First, choose a location for the configuration, for this tutorial we'll create a directory called **singleNode**.

```
mkdir singleNode
cd singleNode
```

Next in the directory place **appsettings.json** and **appsettings.MainChain.MainNet.json** files. An example of **appsettings.json** can be found here. And an example of **appsettings.MainChain.MainNet.json** can be found here

Then you can modify **appsettings.json** file. And the only fields you have to change are the IP and port of your Redis instance :

```
{
  "ConnectionStrings": {
    "BlockchainDb": "redis://192.168.1.70:6379?db=1",
    "StateDb": "redis://192.168.1.70:6379?db=1"
  },
}
```

Replace "192.168.1.70" and 6379 with whatever host your Redis server is on.

2.2.3 Starting the container

Once you have finished downloading the latest version of the AElf image, you can start the container:

```
docker run -it -p 8000:8000 -v <path/to/singleNode>:/opt/aelf-node -w /opt/aelf-node␣
↪aelf/node:latest dotnet /app/AElf.Launcher.dll
```

2.2.4 Access the node's Swagger

You now should have a node that's running, to check this open the browser and enter the address:

```
http://your-ip:8000/swagger/index.html
```

The ip should be localhost if you browser is local.

From here you can try out any of the available API commands on the Swagger page. You can also have a look at the API reference *here*.

2.3 Running multi-nodes with Docker

This section will walk you through the steps for launching three nodes or more by repeating the steps. It is largely based on the section about running a single node (quickstart), so following the fist tutorial is highly recommended. A pre-requisite for this tutorial is to install Docker on your system.

Here's the main points of the tutorial:

- set up some folders that will contain the configuration files for each node.
- use **aelf-command create** to create a key-pair for all nodes.
- modify the **NodeAccount**, **NodeAccountPassword** values in the **appsettings.json**.
- add the key-pair public key as a miner in the **InitialMinerList**.
- launch the nodes with Docker.

Make sure your Redis instance is ready, this tutorial requires three clean instances (we'll use db1, db2 and db3).

2.3.1 Pull AElf Docker image

You can start by opening three terminals, in one of them make sure you have the latest version of the image:

```
docker pull aelf/node
```

Wait for any update to finish.

Create configuration

First, choose a location for the folders, for this tutorial we'll create a directory called **MultiNodeTutorial**, that will become your workspace and navigate inside it.

```
mkdir MultiNodeTutorial
cd MultiNodeTutorial
```

Create three folders in the workspace folder, one for each miner (lets say **miner1**, **miner2** and **miner3**).

```
mkdir miner1 miner2 miner3
```

Next in each of these folders place **appsettings.json** and **appsettings.MainChain.MainNet.json** files. An example of **appsettings.json** can be found here. And an example of **appsettings.MainChain.MainNet.json** can be found here

We'll modify these later in the next steps.

Accounts

Each node will have it's own account, because they will be independent miners.

Generate three accounts, one for each miner, be sure to keep the addresses and the password as well as the password.

```
aelf-command create
```

Be sure to note the public-key and address of all accounts.

Configuration

In this section we will modify the configuration for each miner.

Miners list

Modify each miner's configuration with their respective accounts (**NodeAccount** and **NodeAccountPassword**).

Once this is done you should update config files with public keys, so the configuration for **InitialMinerList** will look something like this in miner1, miner2 and miner3's configuration files:

- MultiNodeTutorial/miner{1, 2 and 3}/appsettings.json:

```
{
  "InitialMinerList" : [
   
↪"0499d3bb14337961c4d338b9729f46b20de8a49ed38e260a5c19a18da569462b44b820e206df8e848185dac6c139f05392
↪",
   
↪"048397dfd9e1035fdd7260329d9492d88824f42917c156aef93fd7c2e3ab73b636f482b8ceb5cb435c556bfa067445a86
↪",
```

(continues on next page)

(continued from previous page)

```
↪"041cc962a51e7bbdd829a8855eca8a03fda708fdf31969251321cb31edadd564bf3c6e7ab31b4c1f49f0f206be81dbe68
↪"
  ],
}
```

Note you need to replace these three public keys with the ones you previously created.

Network

The next section we need to configure is the network options. Following is miner1's configuration of the **Network** section:

```
{
  "Network": {
      "BootNodes": ["x.x.x.x:x"],
      "ListeningPort": 0000,
  },
}
```

Only two options need to be changed for this tutorial, **BootNodes** and **ListeningPort**. The listening port the node will be using to be reachable on the network: other nodes will use this to connect to the node. The boot nodes field is a list of peer addresses that the node will connect to on when it's started. So in order for three nodes to connect to others replace the configurations like following:

- MultiNodeTutorial/miner1/appsettings.json:

```
{
  "Network": {
    "BootNodes": ["192.168.1.70:6801","192.168.1.70:6802"],
    "ListeningPort": 6800
  },
}
```

- MultiNodeTutorial/miner2/appsettings.json:

```
{
  "Network": {
    "BootNodes": ["192.168.1.70:6800","192.168.1.70:6802"],
    "ListeningPort": 6801
  },
}
```

- MultiNodeTutorial/miner3/appsettings.json:

```
{
  "Network": {
    "BootNodes": ["192.168.1.70:6800","192.168.1.70:6801"],
    "ListeningPort": 6802
  },
}
```

Replace "192.168.1.70" with the correct addresses of each node.

Redis

Each node will need it's own database, so in miner2 and miner3 you'll need to change the database number:

- MultiNodeTutorial/miner1/appsettings.json:

```
{
  "ConnectionStrings": {
    "BlockchainDb": "redis://192.168.1.70:6379?db=1",
    "StateDb": "redis://192.168.1.70:6379?db=1"
  },
}
```

- MultiNodeTutorial/miner2/appsettings.json:

```
{
  "ConnectionStrings": {
    "BlockchainDb": "redis://192.168.1.70:6379?db=2",
    "StateDb": "redis://192.168.1.70:6379?db=2"
  },
}
```

- MultiNodeTutorial/miner3/appsettings.json:

```
{
  "ConnectionStrings": {
    "BlockchainDb": "redis://192.168.1.70:6379?db=3",
    "StateDb": "redis://192.168.1.70:6379?db=3"
  },
}
```

Replace "192.168.1.70" and 6379 with whatever host your Redis server is on.

RPC endpoint

The last configuration option we need to change is the RPC endpoint at which the node's API is reachable.

```
{
  "Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "http://*:8000/"
      }
    }
  },
}
```

The example shows that the port is 8000, for miner1 you can keep this value but since we're running this tutorial on a single machine, miner2 and miner3's port must be different, lets say 8001 and 8002.

Start docker - node one

```
docker run -it -p 8000:8000 -p 6800:6800 -v <your/local/keystore/path>:/root/.local/
↪share/aelf/keys -v <path/to/MultiNodeTutorial/miner1>:/opt/aelf-node -w /opt/aelf-
↪node aelf/node:latest dotnet /app/AElf.Launcher.dll
```

Here 8000 will be the API endpoint port and 6800 the listening port and both are mapped in docker. You will need to replace "your/local/keystore/path" to the location of you key store (aelf-command create will show you where your keys are currently stored).

You also need to map the configuration files we created, in the miner1, miner2 and miner3 directories, so replace "path/to/MultiNodeTutorial/miner1" with your local path.

The next step is the same, but with a second container.

Start docker - node two

```
docker run -it -p 8001:8001 -p 6801:6801 -v <your/local/keystore/path>:/root/.local/
↪share/aelf/keys -v <path/to/MultiNodeTutorial/miner2>:/opt/aelf-node -w /opt/aelf-
↪node aelf/node:latest dotnet
```

Here 8001 will be the API endpoint port and 6801 the listening port and both are mapped in docker. Like for the previous node,you will need to replace "your/local/keystore/path" to the location of you key store.

You also need to map the configuration files we created for miner2, in the miner2 directory, so replace "path/to/MultiNodeTutorial/miner2" with your local path.

The next step is the same, but with a third container.

Start docker - node three

```
docker run -it -p 8002:8002 -p 6802:6802 -v <your/local/keystore/path>:/root/.local/
↪share/aelf/keys -v <path/to/MultiNodeTutorial/miner3>:/opt/aelf-node -w /opt/aelf-
↪node aelf/node:latest dotnet
```

Here 8002 will be the API endpoint port and 6802 the listening port and both are mapped in docker. Like for the previous node,you will need to replace "your/local/keystore/path" to the location of you key store.

You also need to map the configuration files we created for miner3, in the miner3 directory, so replace "path/to/MultiNodeTutorial/miner3" with your local path.

2.3.2 Access the node's Swagger

You can now check all nodes with the Swagger interface. Open the following addresses in your browser:

```
http://your-ip:8000/swagger/index.html
http://your-ip:8001/swagger/index.html
http://your-ip:8002/swagger/index.html
```

The ip should be localhost if your browser and docker are running local.

From here you can try out any of the available API commands on the Swagger page. You can also have a look at the API reference *here*.

2.4 QuickStart

2.4.1 Manual build & run the sources

This tutorial is only if you are interested in building the sources and running the node from the build.

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.

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
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*.

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.

Launch and test

Now we build and run the node navigate into the **aelf** directory and build the solution with the following commands:

```
dotnet build AElf.Launcher.csproj --configuration Release
dotnet bin/Release/netcoreapp3.1/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
```

Cleanup

To stop the node you can simply find and kill the process with:

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)
```

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
                  | http://127.0.0.1:8000
      _account    | the instance of an AElf wallet, address
                  | is
                  | 2Ue31YTuB5Szy7cnr3SCEGU2gtGi5uMQBYarYUR...
                  | 5oGin1sys6H
```

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.

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.

Next

This article showed you how to set up the contract's definition and implementation, next, we'll see how to test it.

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
    └── 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.

Next

We've seen how to add the tests for a contract. The next step is to run the node where the contract is deployed to actually be able to interact with the contract from external sources.

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.

Next

We've just seen through this and the previous articles how to use Boilerplate in order to develop and test a smart contract. That said, these articles only show a subset of the possibilities.

The next article will demonstrate how to build a small front-end for the greeter contract.

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);
    }

    // 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);
```

(continues on next page)

(continued from previous page)

```
            });
    };
```

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)
            .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).

Next

This first series of tutorials showed you an end-to-end example of a dApp implemented with Boilerplate. Further tutorials will give more in-depth explanations about some aspect of the contracts.

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 `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.

3.2.2 `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`.

3.2.3 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 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 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" : [

↪"0499d3bb14337961c4d338b9729f46b20de8a49ed38e260a5c19a18da569462b44b820e206df8e848185dac6c139f05392
↪",

↪"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 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 announce election:

```
aelf-command send AElf.ContractNames.Election AnnounceElection '{}' -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 4. User vote election

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

```
aelf-command create

AElf [Info]: Your wallet info is :
```

(continues on next page)

(continued from previous page)

```
AElf [Info]: Mnemonic              : walnut market museum play grunt chuckle hybrid␣
↪accuse relief misery share meadow
AElf [Info]: Private Key           :␣
↪919a220fac2d80e674a256f2367ac840845f344269f4dcdd56d37460de17f947
AElf [Info]: Public Key            :␣
↪04794948de40ffda2a6c884d7e6a99bb8e42b8b96b9ee5cc4545da3a1d5f7725eec93de62ddbfb598ef6f04fe52aa310ac
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":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9
↪","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":
↪"04eed00eb009ccd283798e3862781cebd25ed6a4641e0e1b7d0e3b6b59025040679fc4dc0edc9de166bd630c7255188a9
↪"}'

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

4.5 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 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/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-
↪preview3/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": [
            "testnet-mainchain-1.aelf.io:6800",
            "testnet-mainchain-2.aelf.io:6800"
        ],
        "ListeningPort": 6800,
        "NetAllowed": "",
        "NetWhitelist": []
    }
}
```

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:

```
{
    "CrossChain": {
        "Grpc": {
            "LocalServerPort": 5000,
            "LocalServerHost": "your server ip address",
            "ListeningHost": "0.0.0.0"
        }
    },
}
```

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-preview3
>> cd /opt/aelf-node
>> sh aelf-node.sh start aelf/node:testnet-v1.0.0-preview3
```

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 3.1. For now AElf depends on version 3.1 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-
↪preview3/aelf-v1.0.0-preview3.zip
>> unzip aelf-v1.0.0-preview3.zip
>> mv aelf-v1.0.0-preview3 /opt/aelf-node/
```

Enter the configuration folder and run the node:

```
>> cd /opt/aelf-node
>> dotnet aelf-v1.0.0-preview3/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-preview3), and secondly make sure to compile on a Ubuntu Linux machine (we recommend Ubuntu 18.04.2 LTS) and have dotnet core SDK version 3.1 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:8000
```

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-
↪preview3/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 configuration with the remote information.

```
{
    "CrossChain": {
        "Grpc": {
            "RemoteParentChainServerPort": 5000,
            "LocalServerHost": "you local ip address",
            "LocalServerPort": 5001,
            "RemoteParentChainServerHost": "your mainchain ip address",
            "ListeningHost": "0.0.0.0"
        },
        "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-preview3/aelf-
↪testnet-sidechain1.zip
```

Each side chain has its own P2P network, you can find here some bootnodes that are available:

```
sidechain1 bootnode → ["testnet-sidechain1-1.aelf.io:6800", "testnet-sidechain1-2.
↪aelf.io:6800"]
```

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

CHAPTER 6

Running a side chain

6.1 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.

6.1.1 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 **SideChain-CreationRequest** will determine the type of side chain that is created. For more api details, you can follow the *request side chain creation*.

```
rpc RequestSideChainCreation(SideChainCreationRequest) returns (google.protobuf.
↪Empty) { }
message SideChainCreationRequest {
    int64 indexing_price = 1;
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    string side_chain_token_symbol = 4;
    string side_chain_token_name = 5;
    int64 side_chain_token_total_supply = 6;
    int32 side_chain_token_decimals = 7;
    bool is_side_chain_token_burnable = 8;
    bool is_side_chain_token_profitable = 9;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 10;
    map<string, int32> initial_resource_amount = 11;
}

message SideChainTokenInitialIssue{
```

(continues on next page)

(continued from previous page)

```
    aelf.Address address = 1;
    int64 amount = 2;
}
message ProposalCreated{
    option (aelf.is_event) = true;
    aelf.Hash proposal_id = 1;
}
```

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.

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

message ReleaseSideChainCreationInput {
    aelf.Hash proposal_id = 1;
}

message SideChainCreatedEvent {
    option (aelf.is_event) = true;
    aelf.Address creator = 1;
    int32 chainId = 2;
}
```

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.

6.1.2 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 developer has exclusive use of this side-chain. 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.

6.1.3 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 () => {

    console.log('Starting side chain creation script\n');

    // 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);
        });
```

(continues on next page)

(continued from previous page)

```
    // 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)
{
    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
        });

    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 {
    int64 indexing_price = 1;
```

(continues on next page)

(continued from previous page)

```
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    string side_chain_token_symbol = 4;
    string side_chain_token_name = 5;
    int64 side_chain_token_total_supply = 6;
    int32 side_chain_token_decimals = 7;
    bool is_side_chain_token_burnable = 8;
    bool is_side_chain_token_profitable = 9;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 10;
    map<string, int32> initial_resource_amount = 11;
}

message SideChainTokenInitialIssue{
    aelf.Address address = 1;
    int64 amount = 2;
}

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

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.

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

(continues on next page)

(continued from previous page)

```
});

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

// 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)}`);
```

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:

```
console.log('\n>>>> Approving 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:

```
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);
```

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);

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

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({
```

(continues on next page)

(continued from previous page)

```
        symbol:'ELF',
        spender: crossChainContractAddress,
        amount: 20000
        });

    let approveTransactionResult = await pollMining(approvalResult.TransactionId);
    //console.log(approveTransactionResult);
}

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

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

    let checkAllowanceTxResult = await pollMining(checkAllowanceTx.TransactionId);
    let txReturn = JSON.parse(checkAllowanceTxResult.ReadableReturnValue);

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

const createSideChain = async () => {

    // 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));

    console.log('token contract address: ' + tokenContractAddress);
    console.log('parliament contract address: ' + parliamentContractAddress);
    console.log('cross chain contract address: ' + crossChainContractAddress);

    // 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);
```

(continues on next page)

(continued from previous page)

```
    console.log();

    // 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,
        sideChainTokenDecimals: 8,
        sideChainTokenName: 'SCATokenName',
        sideChainTokenSymbol: 'SCA',
        sideChainTokenTotalSupply: '100000000000000000',
        isSideChainTokenBurnable: true,
        sideChainTokenInitialIssueList: [
            {
                address: '28Y8JA1i2cN6oHvdv7EraXJr9a1gY6D1PpJXw9QtRMRwKcBQMK',
                amount: '1000000000000000'
            }
        ],
        initialResourceAmount: { CPU: 2, RAM: 4, DISK: 512, NET: 1024 },
        isSideChainTokenProfitable: true
    });

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

    // 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');
```

(continues on next page)

(continued from previous page)

```
    console.log('Chain chain created : ');
    console.log(sideChainCreationEvent);
};

createSideChain();
```

6.2 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*.

6.2.1 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": {
```

(continues on next page)

(continued from previous page)

```
    "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.

6.2.2 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 7

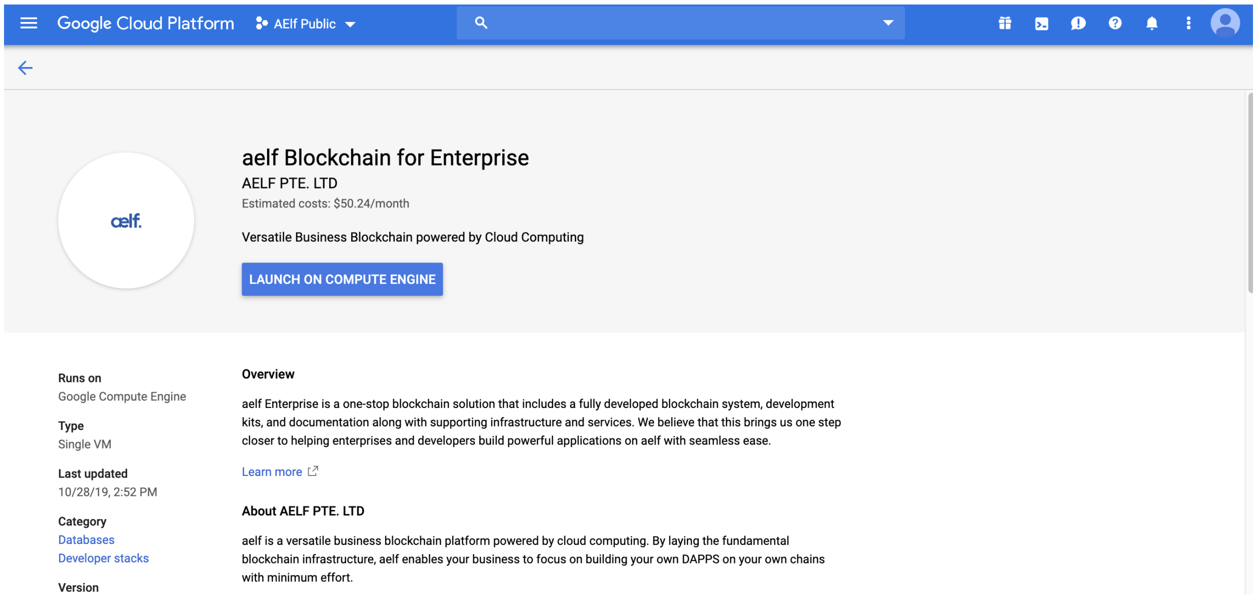
Running AElf on the cloud

This section provides resources for AElf on the cloud.

7.1 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.



aelf Blockchain for Enterprise
AELF PTE. LTD
Estimated costs: $50.24/month

Versatile Business Blockchain powered by Cloud Computing

Runs on
Google Compute Engine

Type
Single VM

Last updated
10/28/19, 2:52 PM

Category
Databases
Developer stacks

Version

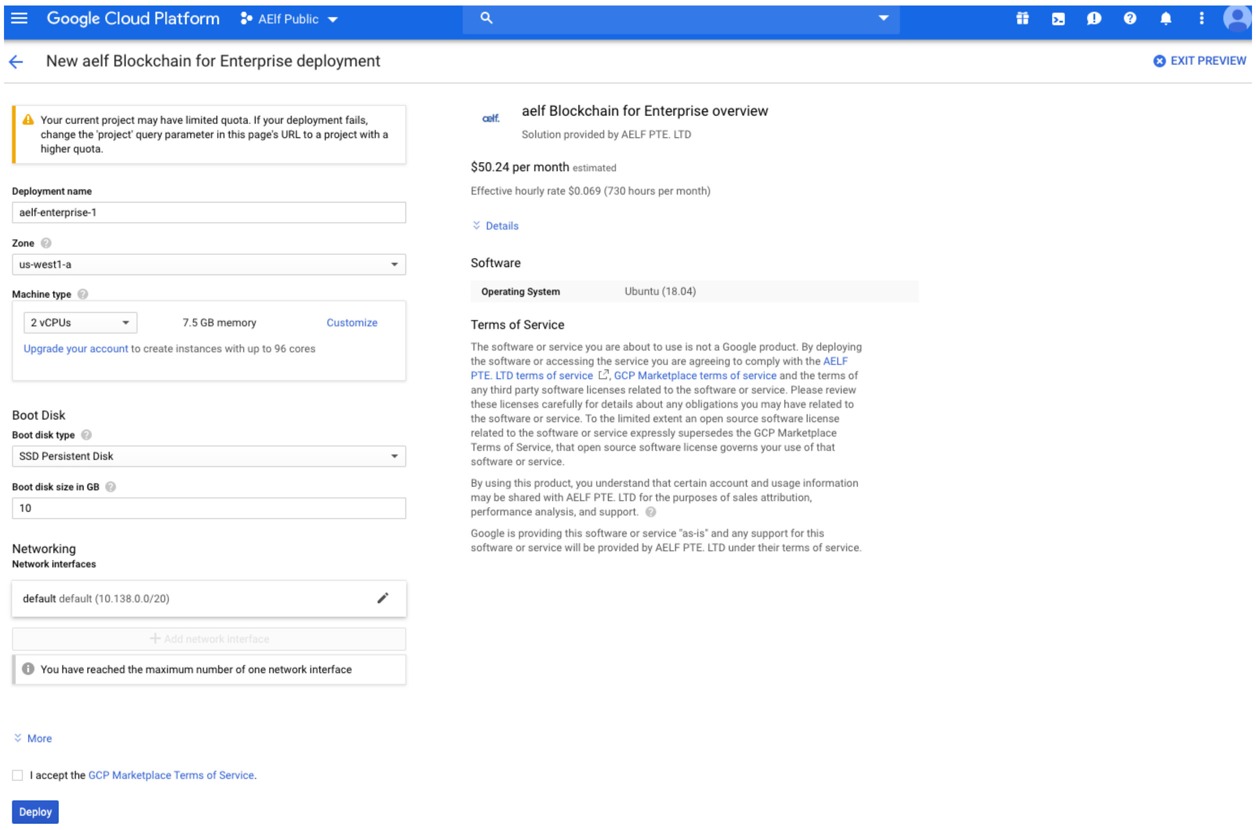
Overview

aelf Enterprise is a one-stop blockchain solution that includes a fully developed blockchain system, development kits, and documentation along with supporting infrastructure and services. We believe that this brings us one step closer to helping enterprises and developers build powerful applications on aelf with seamless ease.

About AELF PTE. LTD

aelf is a versatile business blockchain platform powered by cloud computing. By laying the fundamental blockchain infrastructure, aelf enables your business to focus on building your own DAPPS on your own chains with minimum effort.

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



New aelf Blockchain for Enterprise deployment

Your current project may have limited quota. If your deployment fails, change the 'project' query parameter in this page's URL to a project with a higher quota.

Deployment name: aelf-enterprise-1

Zone: us-west1-a

Machine type: 2 vCPUs, 7.5 GB memory, Customize

Upgrade your account to create instances with up to 96 cores

Boot Disk

Boot disk type: SSD Persistent Disk

Boot disk size in GB: 10

Networking

Network interfaces: default default (10.138.0.0/20)

+ Add network interface

You have reached the maximum number of one network interface

More

I accept the GCP Marketplace Terms of Service.

Deploy

aelf Blockchain for Enterprise overview

Solution provided by AELF PTE. LTD

$50.24 per month estimated

Effective hourly rate $0.069 (730 hours per month)

Details

Software

Operating System	Ubuntu (18.04)

Terms of Service

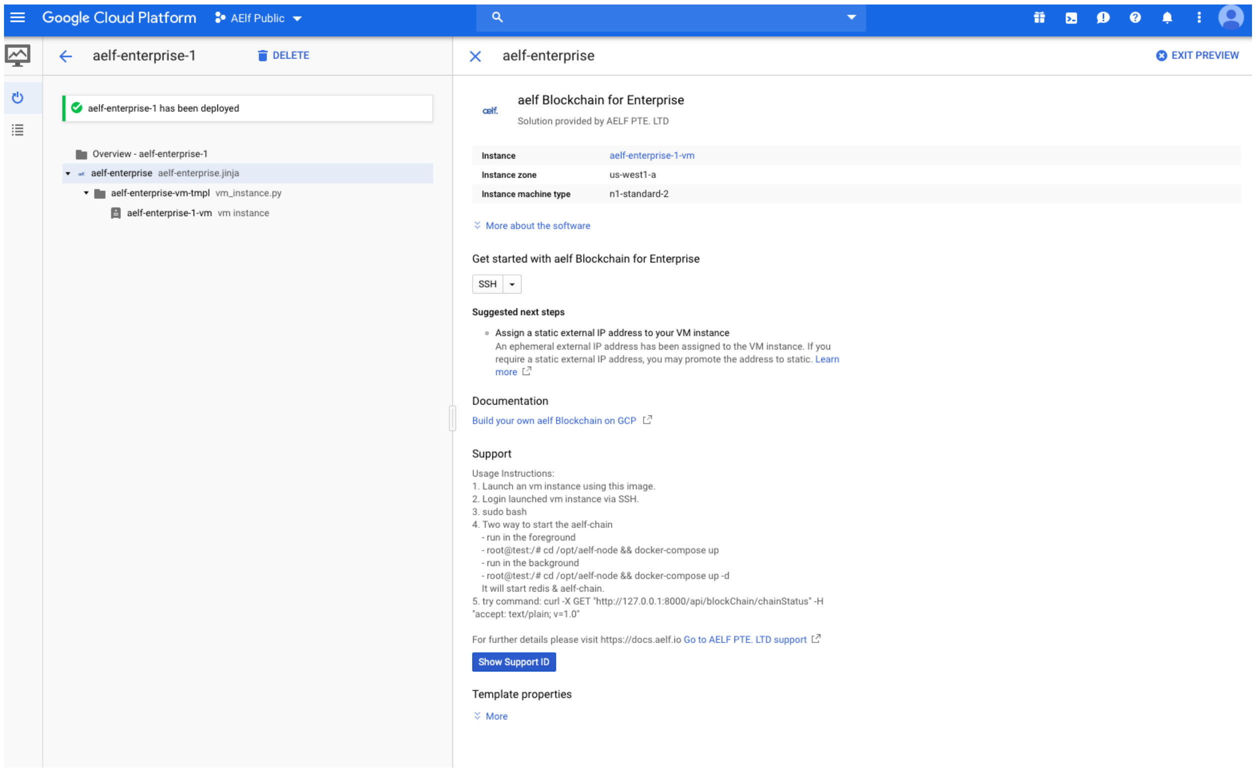
The software or service you are about to use is not a Google product. By deploying the software or accessing the service you are agreeing to comply with the AELF PTE. LTD terms of service, GCP Marketplace terms of service and the terms of any third party software licenses related to the software or service. Please review these licenses carefully for details about any obligations you may have related to the software or service. To the limited extent an open source software license related to the software or service expressly supersedes the GCP Marketplace Terms of Service, that open source software license governs your use of that software or service.

By using this product, you understand that certain account and usage information may be shared with AELF PTE. LTD for the purposes of sales attribution, performance analysis, and support.

Google is providing this software or service "as-is" and any support for this software or service will be provided by AELF PTE. LTD under their terms of service.

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:



aelf-enterprise-1 has been deployed

Overview - aelf-enterprise-1
aelf-enterprise aelf-enterprise.jinja
aelf-enterprise-vm-tmpl vm_instance.py
aelf-enterprise-1-vm vm instance

aelf Blockchain for Enterprise
Solution provided by AELF PTE. LTD

Instance	aelf-enterprise-1-vm
Instance zone	us-west1-a
Instance machine type	n1-standard-2

More about the software

Get started with aelf Blockchain for Enterprise

SSH

Suggested next steps

- Assign a static external IP address to your VM instance
  An ephemeral external IP address has been assigned to the VM instance. If you require a static external IP address, you may promote the address to static. Learn more

Documentation

Build your own aelf Blockchain on GCP

Support

```
Usage Instructions:
1. Launch an vm instance using this image.
2. Login launched vm instance via SSH.
3. sudo bash
4. Two way to start the aelf-chain
  - run in the foreground
  - root@test:/# cd /opt/aelf-node && docker-compose up
  - run in the background
  - root@test:/# cd /opt/aelf-node && docker-compose up -d
  It will start redis & aelf-chain.
5. try command: curl -X GET "http://127.0.0.1:8000/api/blockChain/chainStatus" -H
"accept: text/plain; v=1.0"
```

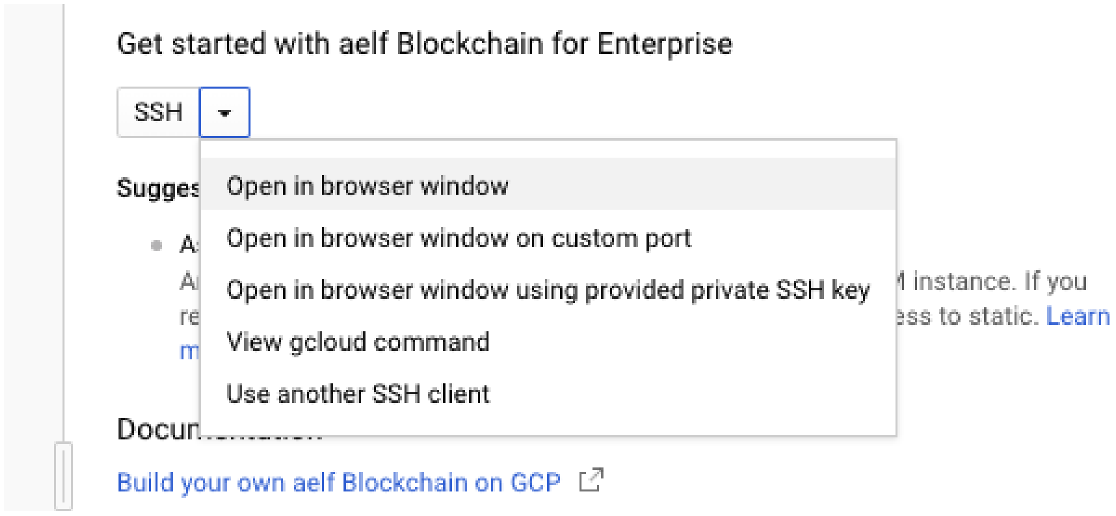
For further details please visit https://docs.aelf.io Go to AELF PTE. LTD support

Show Support ID

Template properties

More

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":

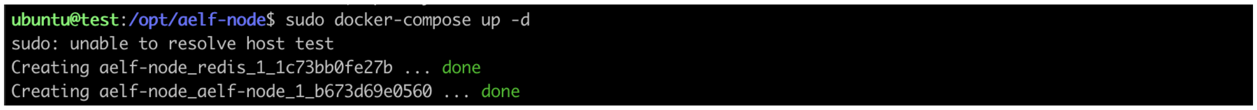

Get started with aelf Blockchain for Enterprise
SSH
Open in browser window
Open in browser window on custom port
Open in browser window using provided private SSH key
View gcloud command
Use another SSH client
Build your own aelf Blockchain on GCP

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`

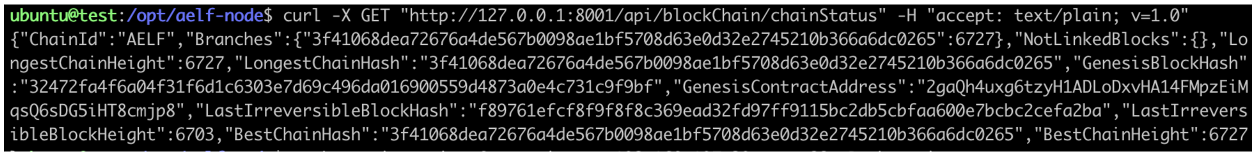
These commands will start redis and an AElf node (the command prints 'done' when finished).



```
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 8

Smart Contract Developing Demos

8.1 Bingo Game

8.1.1 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).

8.1.2 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

8.1.3 Write Contract

Use the code generator to generate contracts and test projects

Open the AElf.Boilerplate.CodeGenerator project in the AElf.Boilerplate solution, and modify the Contents node in appsetting.json under this project:

```
{
  "Contents": [
  {
    "Origin": "AElf.Contracts.HelloWorldContract",
    "New": "AElf.Contracts.BingoContract"
  },
  {
    "Origin": "HelloWorld",
    "New": "Bingo"
  },
  {
    "Origin": "hello_world",
    "New": "bingo"
  }
  ],
}
```

Then run the AElf.Boilerplate.CodeGenerator project. After running successfully, you will see a AElf.Contracts.BingoContract.sln in the same directory as the AElf.Boilerplate.sln is in. After opening the sln, you will see that the contract project and test case project of the Bingo contract have been generated and are included in the new solution.

Define Proto

Based on the API list in the requirements analysis, the bingo_contract.proto file is as follows:

```
syntax = "proto3";
import "aelf/core.proto";
import "aelf/options.proto";
import "google/protobuf/empty.proto";
import "google/protobuf/wrappers.proto";
import "google/protobuf/timestamp.proto";
option csharp_namespace = "AElf.Contracts.BingoContract";
service BingoContract {
    option (aelf.csharp_state) = "AElf.Contracts.BingoContract.BingoContractState";

    // Actions
    rpc Register (google.protobuf.Empty) returns (google.protobuf.Empty) {
    }
    rpc Play (google.protobuf.Int64Value) returns (google.protobuf.Int64Value) {
    }
    rpc Bingo (aelf.Hash) returns (google.protobuf.BoolValue) {
    }
    rpc Quit (google.protobuf.Empty) returns (google.protobuf.Empty) {
    }

    // Views
    rpc GetAward (aelf.Hash) returns (google.protobuf.Int64Value) {
        option (aelf.is_view) = true;
    }
    rpc GetPlayerInformation (aelf.Address) returns (PlayerInformation) {
        option (aelf.is_view) = true;
    }
}
message PlayerInformation {
    aelf.Hash seed = 1;
    repeated BoutInformation bouts = 2;
    google.protobuf.Timestamp register_time = 3;
}
message BoutInformation {
    int64 play_block_height = 1;
    int64 amount = 2;
    int64 award = 3;
    bool is_complete = 4;
    aelf.Hash play_id = 5;
    int64 bingo_block_height = 6;
}
```

Contract Implementation

Here only talk about the general idea of the Action method, specifically need to turn the code:

https://github.com/AElfProject/aelf-boilerplate/blob/dev/chain/contract/AElf.Contracts.BingoGameContract/BingoGameContract.cs

Register & Quit

Register

1. Determine the Seed of the user, Seed is a hash value, participating in the calculation of the random number, each user is different, so as to ensure that different users get different results on the same height;

2. Record the user's registration time.

QuitJust delete the user's information.

Play & Bingo

Play

1. Use TransferFrom to deduct the user's bet amount;
2. At the same time add a round (Bount) for the user, when the Bount is initialized, record three messages 1.PlayId, the transaction Id of this transaction, is used to uniquely identify the Bout (see BoutInformation for its data structure in the Proto definition); 2.AmountRecord the amount of the bet 3.Record the height of the block in which the Play transaction is packaged.

Bingo

1. Find the corresponding Bout according to PlayId, if the current block height is greater than PlayBlockHeight + number of nodes * 8, you can get the result that you win or lose;
2. Use the current height and the user's Seed to calculate a random number, and then treat the hash value as a bit Array, each of which is added to get a number ranging from 0 to 256.
3. Whether the number is divisible by 2 determines the user wins or loses;
4. The range of this number determines the amount of win/loss for the user, see the note of GetKind method for details.

8.1.4 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 9

Consensus

9.1 Overview

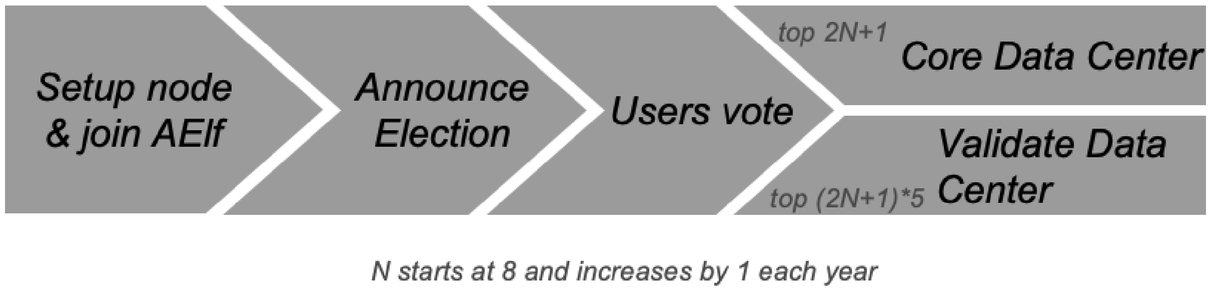
The process of reaching consensus is an essential part of every blockchain, since its what determines which transactions get included in the block and in what order. A stable and efficient Block formation mechanism is the foundation of the AElf system. The operation and maintenance of AElf is more complicated 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**.

9.1.1 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.

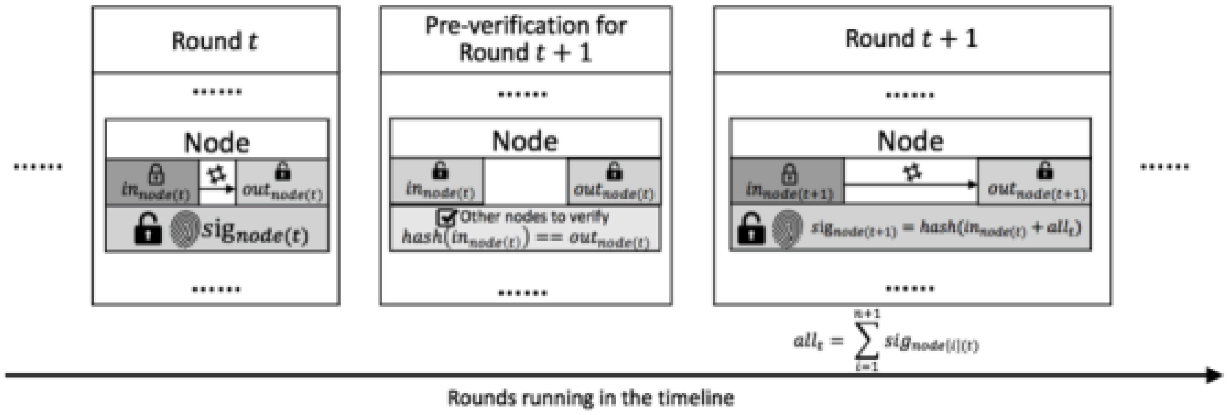
9.1.2 Validate Data Center

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.

9.2 AEDPoS Process

9.2.1 Round

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


Round t
Pre-verification for Round $t+1$
Round $t+1$
Node
$in_{node(t)}$
$out_{node(t)}$
$sig_{node(t)}$
Other nodes to verify
$hash(in_{node(t)}) == out_{node(t)}$
$in_{node(t+1)}$
$out_{node(t+1)}$
$sig_{node(t+1)} = hash(in_{node(t)} + all_t)$
$all_t = \sum_{i=1}^{n+1} sig_{node[i](t)}$
Rounds running in the timeline

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)**.

9.2.2 Main Processes

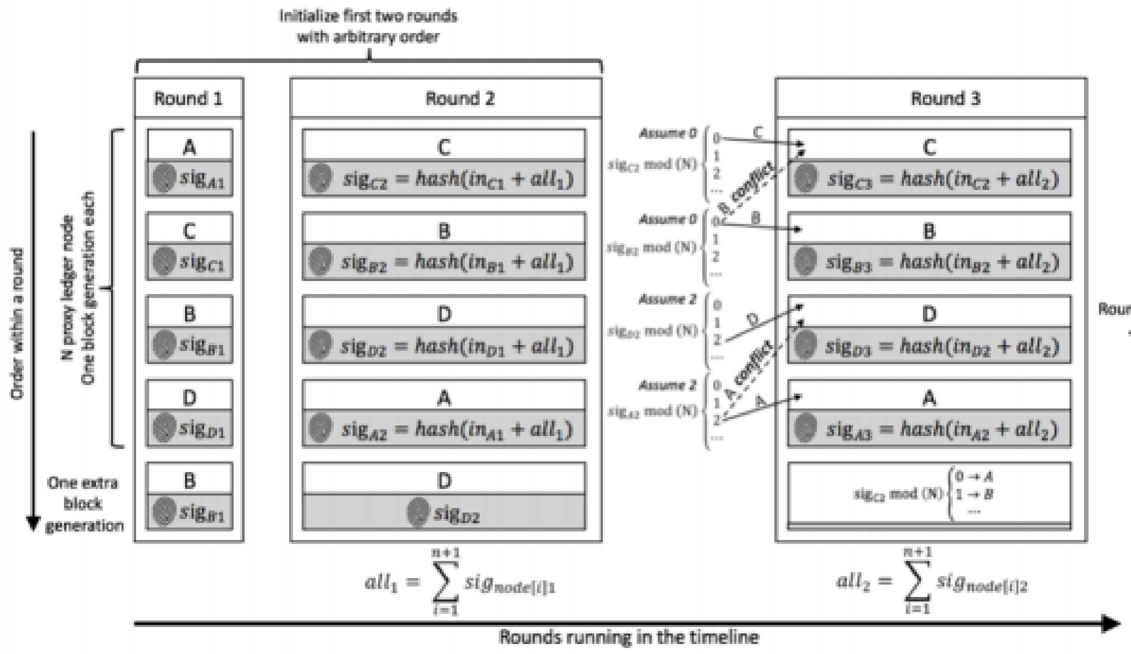
Pre-Verification

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.



Initialize first two rounds with arbitrary order
Round 1
Round 2
Round 3
Order within a round
N proxy ledger node One block generation each
One extra block generation
sig_{A1}
sig_{C1}
sig_{B1}
sig_{D1}
$sig_{C2} = hash(in_{C1} + all_1)$
$sig_{B2} = hash(in_{B1} + all_1)$
$sig_{D2} = hash(in_{D1} + all_1)$
$sig_{A2} = hash(in_{A1} + all_1)$
sig_{D2}
$sig_{C3} = hash(in_{C2} + all_2)$
$sig_{B3} = hash(in_{B2} + all_2)$
$sig_{D3} = hash(in_{D2} + all_2)$
$sig_{A3} = hash(in_{A2} + all_2)$
Assume 0
Assume 2
B conflict
A conflict
Round 4, 5, 6
$all_1 = \sum_{i=1}^{n+1} sig_{node[i]1}$
$all_2 = \sum_{i=1}^{n+1} sig_{node[i]2}$
Rounds running in the timeline

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.

9.3 Irreversible Block

todo

CHAPTER 10

Network

10.1 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.

10.2 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…

10.2.1 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.

10.2.2 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.

10.2.3 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.

10.3 3. Protocol

Each node implements the network interface protocol defined by AElf to ensure normal operation and data synchronization between nodes.

10.3.1 3.1 Connection

3.1.1 DoHandshake Interface

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 Interface

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 {
}
```

10.3.2 3.2 Broadcasting

3.2.1 BlockBroadcastStream Interface

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:

3.2.2 TransactionBroadcastStream Interface

Interface TransactionBroadcastStream used to receive other nodes forward transaction information.

```
rpc TransactionBroadcastStream (stream aelf.Transaction) returns (VoidReply) {}
```

3.2.3 AnnouncementBroadcastStream Interface

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.

3.2.4 LibAnnouncementBroadcastStream Interface

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.

10.3.3 3.3 Block Request

3.3.1 RequestBlock Interface

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.

3.3.2 RequestBlocks Interface

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.

10.3.4 3.4 Peer Management

3.4.1 Ping Interface

Interface Ping is used between nodes to verify that each other's network is available.

```
rpc Ping (PingRequest) returns (PongReply) {}
```

```
message PingRequest {
}
```

```
message PongReply {
}
```

3.4.2 CheckHealth Interface

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 11

Address

11.1 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.

11.2 User Address

A User Address is generated by an instance of a implementation of interface *IAElfAsymmetricCipherKeyPair*. We call this instance as a key pair. 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 passwrod. 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" (here for more on this):

```
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.

11.3 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);
}
```

11.4 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 12

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.

12.1 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.

12.2 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;
}
```

12.3 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 theres 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.

12.3.1 From

The address of the sender of a transaction.

Note that the **From** is not currently useful because we derive it from the signature.

12.3.2 To

The address of the contract when calling a contract.

12.3.3 MethodName

The name of a method in the smart contract at the **To** address.

12.3.4 Params

The parameters to pass to the aforementioned method.

12.3.5 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);
```

12.3.6 RefBlockNumber & RefBlockPrefix

These are security measures, to know when the transaction broadcasted and whether this transaction has expired.

12.3.7 Transaction Id

The unique identity of a transaction.

Note that if the sender broadcasted several transaction with same parameters at the same time, the Transaction Id of these transactions will be the same, thus these transactions will be regarded as one transaction but broadcasted several times.

12.4 Transaction Execution

todo

CHAPTER 13

Core

13.1 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.

13.1.1 Frameworks and libraries:

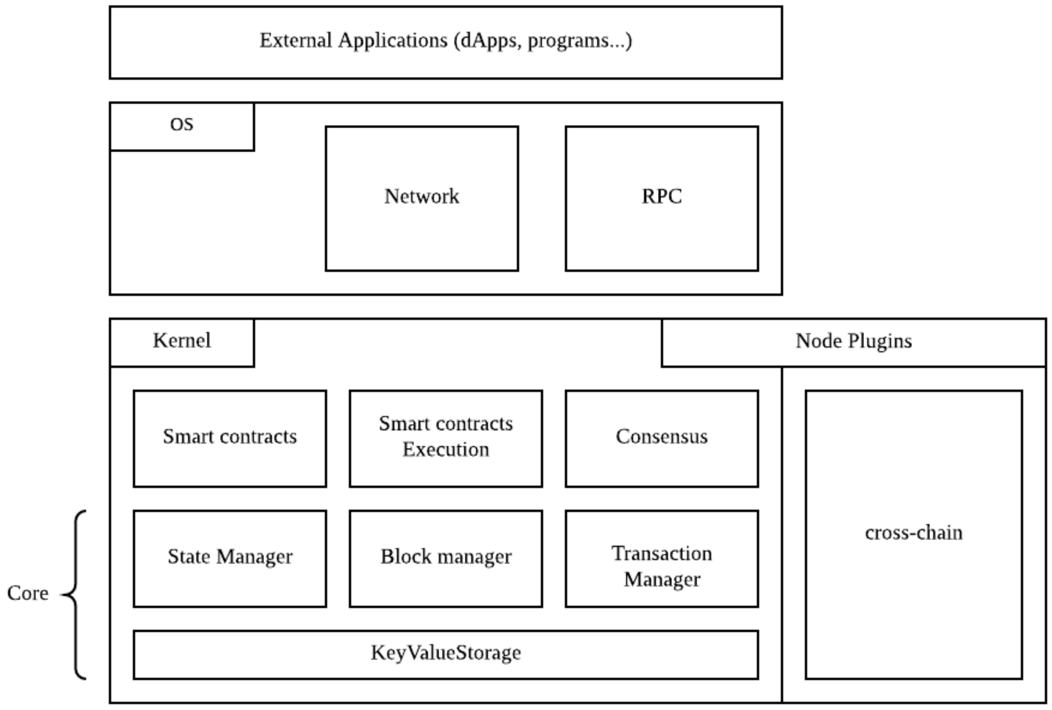
The main programming language used to code and build AElf is C# and is built with the dotnet core framework. It's a choice that was made due to the excellent performances observed with the framework. Dotnet core also comes with the benefit of being cross platform, at least for the three main ones that 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.

13.2 Design principles:



{.align-center width="300px"}

The above diagram shows the conceptual structure of the node and the separation between OS and Kernel.

13.2.1 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.

13.2.2 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.

13.2.3 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.

13.2.4 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 14

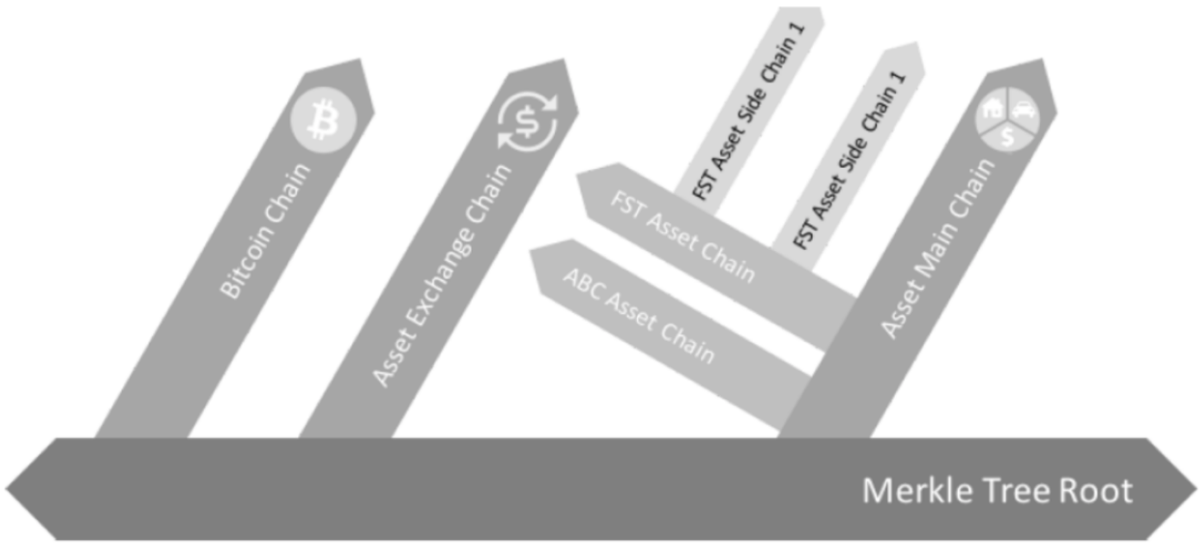
Cross Chain

14.1 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.

14.2 Overview

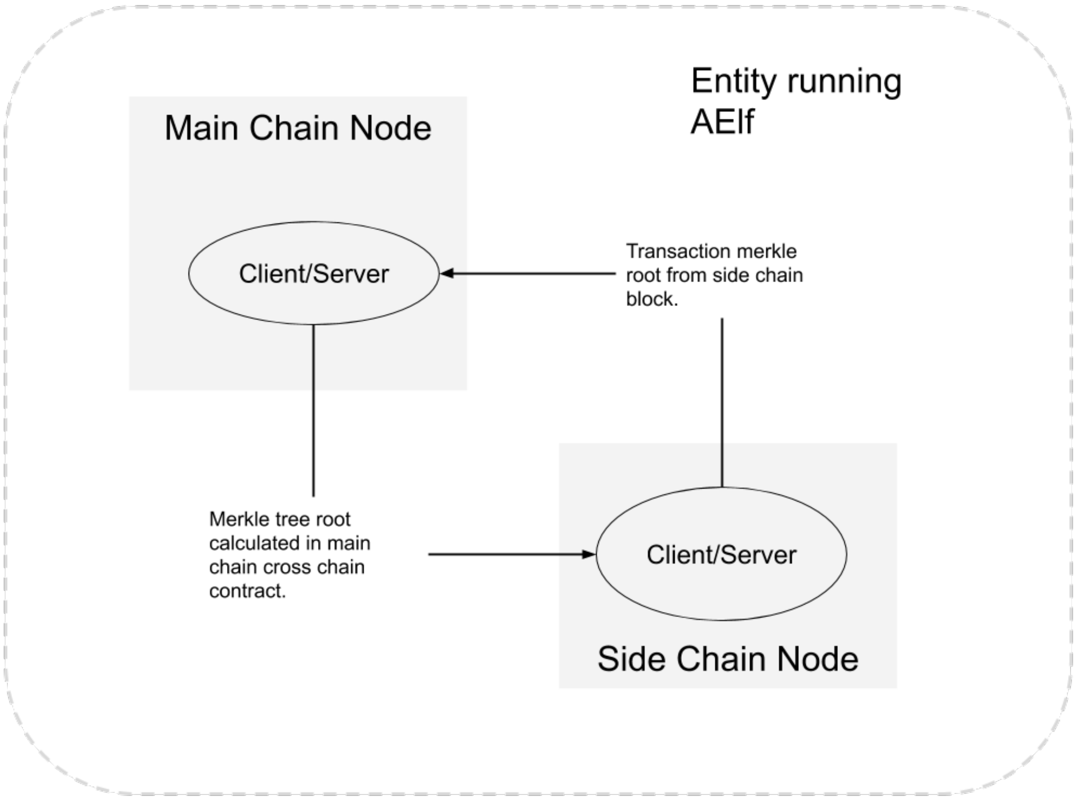
Conceptually a side chain node and main chain node are similar, they are both independent blockchains, with their own peer-to-peer network and possibly their own ecosystem. It is even possible to have this setup on multiple levels. In terms of peer-to-peer networks, all side chains work in parallel to 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.

14.2.1 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.


Entity running AElf
Main Chain Node
Client/Server
Transaction merkle root from side chain block.
Merkle tree root calculated in main chain cross chain contract.
Client/Server
Side Chain Node

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.

The next section describes what happens once the side chain node has been started.

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

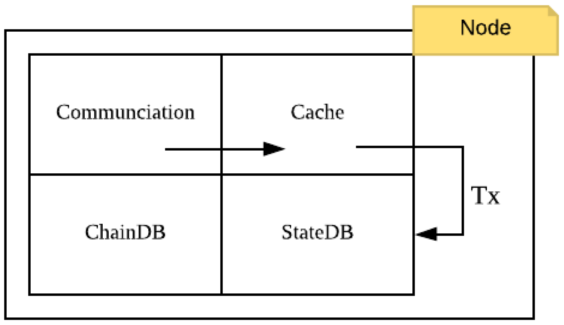
For effective indexing, a cache layer is used to store cross chain data received from remote nodes, and make it available and correct. Cross chain data is cached by chain id and block height with a count limit. The cache layer can give the data if cached when the node needs it. So cache layer decouples 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:



14.3 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.

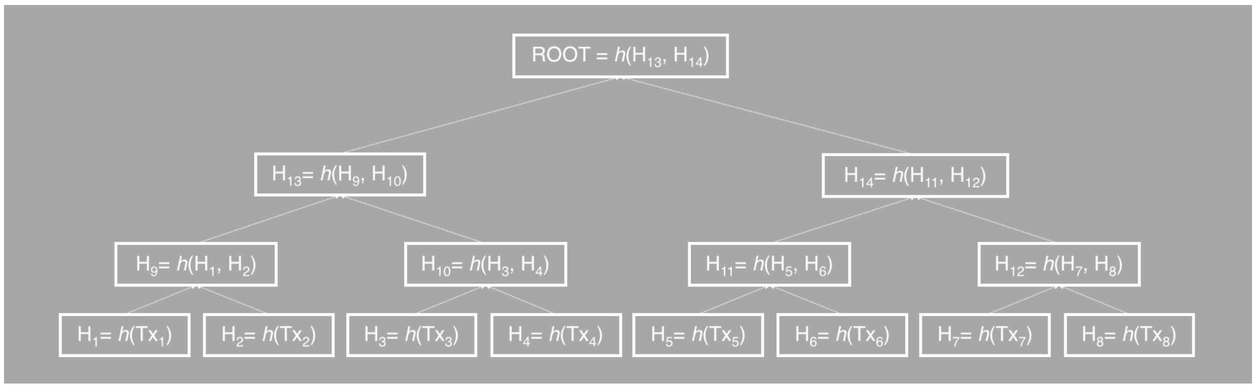
14.3.1 Indexing

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.

14.3.2 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.

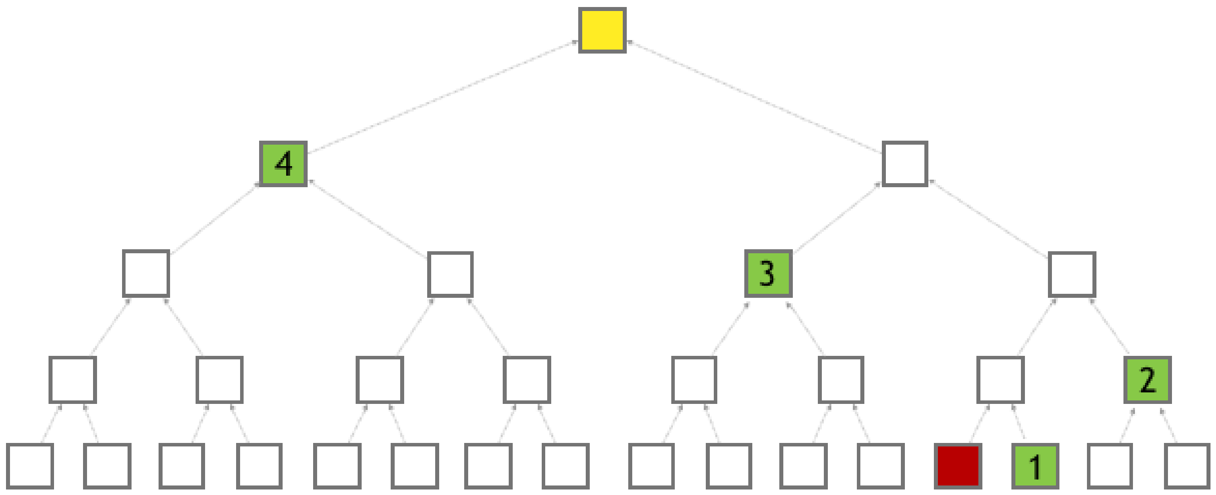


14.3.3 Merkle roots

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.

14.3.4 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.



14.4 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.

14.4.1 Send a transaction

Any transaction with status `Mined` can be verified, the only pre-condition is that the transaction was indexed.

14.4.2 Verify the transaction

There's basically two scenarios that can be considered:

- verifying a main-chain transaction on a side chain.
- verifying a side-chain transaction on the main-chain or another side-chain.

```
rpc VerifyTransaction (VerifyTransactionInput) returns (google.protobuf.BoolValue) {
  option (aelf.is_view) = true;
}

message VerifyTransactionInput {
  aelf.Hash transaction_id = 1;
  aelf.MerklePath path = 2;
  sint64 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 transaction from main-chain on the side-chain

Verifying a transaction sent on the main-chain 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 transaction from side-chain on the main-chain or another side-chain

Before you can verify a transaction sent on a side-chain, on the main-chain or on another side-chain, you need use another contract api **GetBoundParentChainHeightAndMerklePathByHeight** from `cross chain contract`, on the source chain, in which the transaction was packed.

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 main-chain (or the other side-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**.
  - next, use **GetBoundParentChainHeightAndMerklePathByHeight** and 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.

14.5 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

14.5.1 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;
    aelf.Address issuer = 5;
    bool is_burnable = 6;
    int32 issue_chain_id = 7;
    bool is_profitable = 8;
}
```

- Create token `FOO` on chain `B`.

Send transaction `tx_2` 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;
}
```

14.5.2 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

14.5.3 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.SInt64Value) returns␣
↪(CrossChainMerkleProofContext) {
    option (aelf.is_view) = true;
}

message CrossChainMerkleProofContext {
    int64 bound_parent_chain_height = 1;
```

(continues on next page)

(continued from previous page)

```
    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**
  - for the case of transfer from main chain to side chain: for this you just need the merkle path from the main chain's web api with the **GetMerklePathByTransactionIdAsync** method (**CrossChainTransfer** transaction ID as input).
  - for the case of transfer from side chain to side chain or from side chain to main chain: for this you also need to get the merkle path from the source node (side chain here). But you also have to complete this merkle path with **GetBoundParentChainHeightAndMerklePathByHeight** with the **CrossChainTransfer** transaction's block height (concat the merkle path nodes). The nodes are in the **merkle_path_from_parent_chain** field of the **CrossChainMerkleProofContext** object.

CHAPTER 15

Smart contract

15.1 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 defination 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.

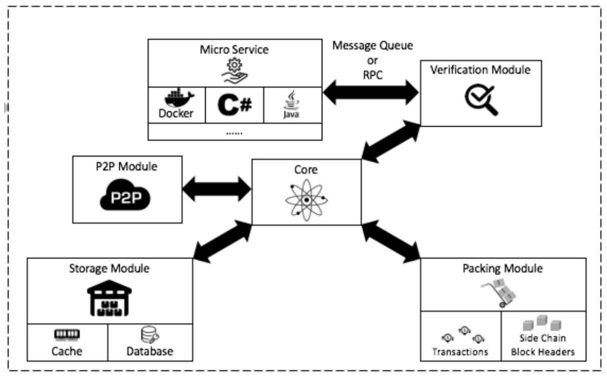
Transactions trigger the logic contained in smart contracts. If a user of the blockchain wants to modify some state, he needs to build a transaction that will call a specific method on some contract. When the transaction is included in a block and this block is executed, the modifications will be executed.

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.

15.1.1 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

15.1.2 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**.

15.1.3 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.

15.1.4 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.

15.2 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.

15.2.1 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.

15.3 Smart contract events

15.3.1 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;
    Address from = 1;
    Address to = 2;
    string symbol = 3;
    sint64 amount = 4;
}
```

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.

15.4 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.

15.5 Development Requirements and Restrictions

There are several requirements and restrictions for a contract to be deployable that are classified into below categories:

15.5.1 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>netstandard2.0</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.

15.5.2 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.

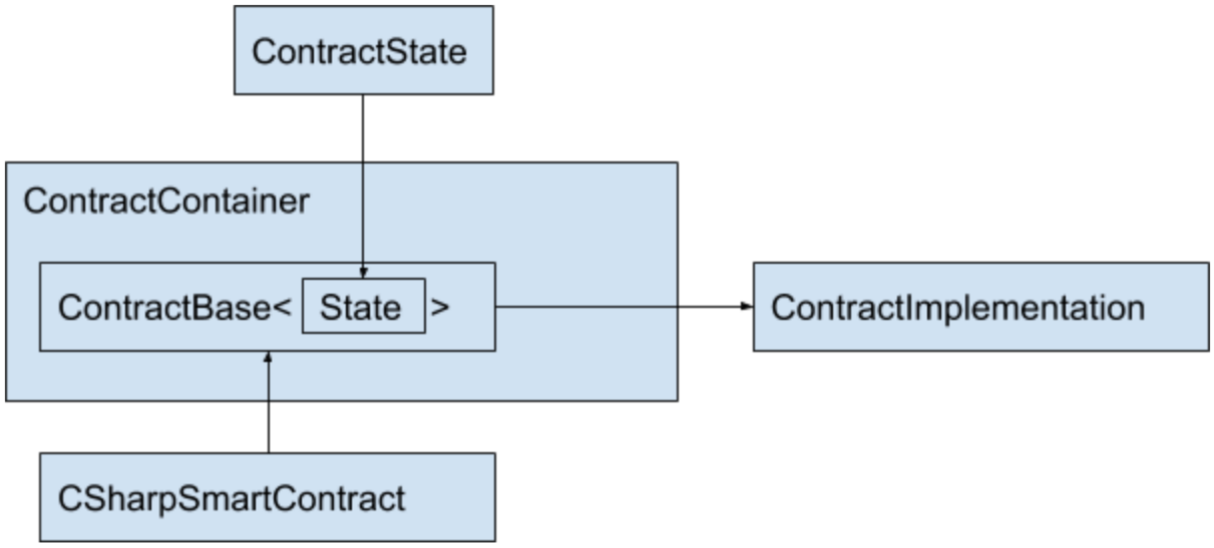


Fig. 1: Contract Class Structure

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`

15.5.3 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	Size Limit	Threshold
`byte`	By Size	4 Mb
`short`	By Size	4 Mb
`int`	By Size	4 Mb
`long`	By Size	4 Mb
`ushort`	By Size	4 Mb
`uint`	By Size	4 Mb
`ulong`	By Size	4 Mb
`decimal`	By Size	4 Mb
`char`	By Size	4 Mb
`string`	By Size	128
`Type`	By Size	5
`Object`	By Size	5
`FileDescriptor`	By Count	10
`GeneratedClrTypeInfo`	By Count	100

15.5.4 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.
- 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 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`

CHAPTER 16

AELF API 1.0

16.1 Chain API

16.1.1 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

16.1.2 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

16.1.3 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

16.1.4 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

16.1.5 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

16.1.6 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

16.1.7 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

16.1.8 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

16.1.9 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

16.1.10 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

16.1.11 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

16.1.12 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

16.1.13 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

16.1.14 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

16.1.15 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

16.1.16 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

16.1.17 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

16.2 Net API

16.2.1 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

16.2.2 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

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

16.2.3 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

Produces

- `text/plain; v=1.0`
- `application/json; v=1.0`
- `text/json; v=1.0`
- `application/x-protobuf; v=1.0`

Tags

- Net

16.2.4 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

- Net

16.2.5 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*

BlockHeaderDto

Name	Schema
Bloom *optional*	string
ChainId *optional*	string
Extra *optional*	string
Height *optional*	integer (int64)
MerkleTreeRootOfTransactions *optional*	string
MerkleTreeRootOfWorldState *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

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

CHAPTER 17

Chain SDK

17.1 aelf-sdk.js - AELF JavaScript API

17.1.1 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

17.1.2 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'));
```

17.1.3 Examples

You can also see full examples in `./examples`;

1.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'));
```

2.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'));
```
~~~

17.1.4 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'*

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

13.addPeer

Attempts to add a node to the connected network nodes

14.removePeer

Attempts to remove a node from the connected network nodes

17.1.5 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`

17.1.6 AElf.pbjs

The reference to protobuf.js, read the documentation to see how to use.

17.1.7 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
```

17.1.8 AElf.version

```
import AElf from 'aelf-sdk';
AElf.version // eg. 3.2.23
```

17.1.9 Requirements

- Node.js
- NPM

17.1.10 Support

17.1.11 About contributing

Read out [contributing guide]

17.1.12 About Version

https://semver.org/

17.2 aelf-sdk.cs - AELF C# API

This C# library helps in the communication with an AElf node. You can find out more **here**.

17.3 aelf-sdk.go - AELF Go API

This Go library helps in the communication with an AElf node. You can find out more **here**.

17.4 aelf-sdk.java - AELF Java API

This Java library helps in the communication with an AElf node. You can find out more **here**.

17.5 aelf-sdk.php - AELF PHP API

This PHP library helps in the communication with an AElf node. You can find out more **here**.

17.6 aelf-sdk.py - AELF Python API

This Python library helps in the communication with an AElf node. You can find out more **here**.

CHAPTER 18

C# reference

18.1 AElf.Sdk.CSharp

18.1.1 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.

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.

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.

DecryptMessage(senderPublicKey,cipherMessage) `method`

Summary

Decrypts a message with the given public key.

Returns

The decrypted message.

Parameters

Name	Type	Description
senderPublicKey	System.Byte[]	The public key that encrypted the message.
cipherMessage	System.Byte[]	The encrypted message.

EncryptMessage(receiverPublicKey,plainMessage) `method`

Summary

Encrypts a message with the given public key.

Returns

The encrypted message.

Parameters

Name	Type	Description
receiverPublicKey	System. Byte[]	The receivers public key.
plainMessage	System. Byte[]	The non encrypted message.

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.

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.		

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.
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.

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. ISmartContractBridge-Context	An instance of ISmartContractBridge-Context
virtualAd-dress	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. ISmartContractBridge-Context	An instance of ISmartContractBridge-Context
virtualAd-dress	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. ISmartContractBridgeContext	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. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext

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. ISmartContractBridgeContext	An instance of ISmartContractBridgeContext
address	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. ISmartContract-BridgeContext	An instance of ISmartContractBridgeContext
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
method-Name	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. ISmartContract-BridgeContext	An instance of ISmartContractBridgeContext
toAddress	AElf.Types.Address	The address of the contract you're seeking to interact with.
method-Name	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.

18.2 AElf.CSharp.Core

18.2.1 Contents

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 19

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**).

19.1 Association Contract

19.1.1 Actions

CreateOrganization

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
}

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

Creates an organization and returns its address.

- **CreateOrganizationInput**
  - **organizer member list**: initial organization members.
  - **proposal release threshold**: the threshold for releasing the proposal.
  - **proposer white list**: proposer whitelist.
- **OrganizationMemberList**
  - **organization members**: addresses of the initial organization members.
- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the value for the minimum approval threshold.
  - **maximal rejection threshold**: the value for the maximal rejection threshold.
  - **maximal abstention threshold**: the value for the maximal abstention threshold.
  - **minimal vote threshold**: the value for the minimal vote threshold.
- **ProposerWhiteList**
  - **proposers**: address of the proposers.
- **Returns**
  - **value**: the address of newly created organization.
- **Events**
  - **OrganizationCreated**
    - **organization address**: the address of newly created organization

CreateOrganizationBySystemContract

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

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

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

message OrganizationMemberList {
```

(continues on next page)

(continued from previous page)

```
    repeated aelf.Address organization_members = 1;
}

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

Creates an organization by system contract and returns its address.

- **CreateOrganizationBySystemContractInput**
  - **CreateOrganizationInput**: paramenters of creating organization.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

note: *for CreateOrganizationInput see CreateOrganization*

- **Returns**
  - **value**: the address of newly created organization.
- **Events**
  - **OrganizationCreated**

note: *for OrganizationCreated see CreateOrganization*

ChangeOrganizationMember

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

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

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

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

- **OrganizationMemberList**
  - **organization_members**: the new members.
- **OrganizationMemberChanged**
  - **organization_address**: the organization address.
  - **organization_member_list**: the new member list.

19.1.2 ACS3 specific methods

CreateProposal

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

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

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

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

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

Approve

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

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

This method is called to approve the specified proposal.

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

Reject

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

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

This method is called to reject the specified proposal.

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

note: *for ReceiptCreated see Approve*

Abstain

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

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

This method is called to abstain from the specified proposal.

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

note: *for ReceiptCreated see Approve*

Release

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

This method is called to release the specified proposal.

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

note: *for ReceiptCreated see Approve*

ChangeOrganizationThreshold

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

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

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

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

note: *for ProposalReleaseThreshold see CreateOrganization*

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

ChangeOrganizationProposerWhiteList

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

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

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

(continues on next page)

(continued from previous page)

```
    ProposerWhiteList proposer_white_list = 2;
}
```

This method overrides the list of whitelisted proposers.

note: *for ProposerWhiteList see CreateOrganization*

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

CreateProposalBySystemContract

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

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

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

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**
  - **CreateProposalInput**: the parameters of creating proposal.
  - **origin proposer**: the actor that trigger the call.

note: *for CreateProposalInput see CreateProposal*

- **Returns**
  - **value**: newly created organization address.
- **Events**
  - **ProposalCreated**
    - **proposal_id**: id of the created proposal.

ClearProposal

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

Removes the specified proposal.

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

ValidateOrganizationExist

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

Checks the existence of an organization.

- **Address**: organization address to be checked.
- **Returns**
  - **value**: indicates whether the organization exists.

19.1.3 View methods

GetOrganization

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

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

Returns the organization with the specified address.

- **Address**: organization address.
- **Returns**
  - **member list**: original members of this organization.
  - **ProposalReleaseThreshold**: the threshold of release the proposal.
  - **ProposerWhiteList**: the proposals in the whitelist.
  - **address**: the organizations address.
  - **hash**: the organizations id.

  note: *for ProposalReleaseThreshold and ProposerWhiteList see CreateOrganization*

CalculateOrganizationAddress

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

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

Calculates with input and returns the organization address.

note: *for CreateOrganizationInput and ProposerWhiteList see CreateOrganization*

- **Returns**

  - **value**: organization address.

GetProposal

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

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

Get the proposal with the given id.

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

ValidateProposerInWhiteList

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

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

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**

  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.
- **Returns**
  - **value**: indicates whether the proposer is whitelisted.

19.2 Referendum Contract

19.2.1 Actions

CreateOrganization

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

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

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

Creates an organization and returns its address.

- **CreateOrganizationInput**
  - **token symbol**: the token used during proposal operations.
  - **proposal release threshold**: the threshold for releasing the proposal.
  - **proposer white list**: proposer whitelist.
- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the value for the minimum approval threshold.
  - **maximal rejection threshold**: the value for the maximal rejection threshold.
  - **maximal abstention threshold**: the value for the maximal abstention threshold.
  - **minimal vote threshold**: the value for the minimal vote threshold.
- **ProposerWhiteList**
  - **proposers**: address of the proposers.
- **Returns**
  - **Address**: newly created organization address.
- **Events**
  - **OrganizationCreated**
    * **organization address**: the address of newly created organization.

CreateOrganizationBySystemContract

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

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

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

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

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

- **CreateOrganizationBySystemContractInput**
  - **CreateOrganizationInput**: the parameters of creating a organization.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

note: *for CreateOrganizationInput see CreateOrganization*

- **Returns**
  - **Address**: newly created organization address.
- **Events**
  - **OrganizationCreated**
    * **organization address**: the address of newly created organization

ReclaimVoteToken

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

Used to unlock the tokens that where used for voting.

- **Hash**: proposal id.

19.2.2 ACS3 specific methods

CreateProposal

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

message CreateProposalInput {
```

(continues on next page)

(continued from previous page)

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

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

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

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

Approve

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

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

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

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

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

Reject

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

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

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

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

note: *for ReferendumReceiptCreated see Approve*

Abstain

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

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

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

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

- **Events**
  - **ReferendumReceiptCreated**

note: *for ReferendumReceiptCreated see Approve*

Release

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

This method is called to release the specified proposal.

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

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

ChangeOrganizationThreshold

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

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

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

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

note: *for ProposalReleaseThreshold see CreateOrganization*

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

ChangeOrganizationProposerWhiteList

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

message ProposerWhiteList {
```

(continues on next page)

(continued from previous page)

```
    repeated aelf.Address proposers = 1;
}

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

This method overrides the list of whitelisted proposers.

- **ProposerWhiteList**
  - **proposers**: the new value for the proposer whitelist.
- **Events**
  - **OrganizationWhiteListChanged**
    - **organization_address**: the organization address.
    - **proposer_white_list**: the new proposer whitelist.

CreateProposalBySystemContract

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

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

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

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**
  - **CreateProposalInput**: the parameters of creating a proposal.
  - **origin proposer**: the actor that trigger the call.

note: *for CreateProposalInput see CreateProposal*

- **Returns**
  - **Address**: newly created organization address.
- **Events**
  - **ProposalCreated**
    - **proposal_id**: id of the created proposal.

ClearProposal

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

Removes the specified proposal.

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

ValidateOrganizationExist

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

Checks the existence of an organization.

- **Address**: organization address to be checked.
- **Returns**
  - **BoolValue**: indicates whether the organization exists.

19.2.3 View methods

GetOrganization

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

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

Returns the organization with the provided organization address.

- **Address**: organization address.
- **Returns**
  - **ProposalReleaseThreshold**: the proposal releash threshold.
  - **token**: token used for proposal operations.
  - **organization address**: organization address.
  - **organization hash**: organization id.
  - **ProposerWhiteList**: proposals in whitelist.

note: *for ProposalReleaseThreshold and ProposerWhiteList see CreateOrganization*

CalculateOrganizationAddress

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

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

Calculates with input and returns the organization address.

note: *for CreateOrganizationInput see CreateOrganization*

- **Returns**
  - **Address**: organization address.

GetProposalVirtualAddress

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

Get virtual address for the proposal.

- **Hash**: id of the proposal.
- **Returns**
  - **Address**: the virtual address for proposal.

GetProposal

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

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

Gets the proposal with the given id.

- **Hash**: proposal id.
- **Returns**
  - **proposal id**: id of the proposal.
  - **method name**: the method that this proposal will call when being released.
  - **to address**: the address of the target contract.
  - **params**: the parameters of the release transaction.

- **expiration**: the date at which this proposal will expire.
- **organization address**: address of this proposals organization.
- **proposer**: address of the proposer of this proposal.
- **to be release**: indicates if this proposal is releasable.
- **approval count**: locked token amount for approval.
- **rejection count**: locked token amount for rejection.
- **abstention count**: locked token amount for abstention.

ValidateProposerInWhiteList

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

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

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**
  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.
- **Returns**
  - **BoolValue**: indicates whether the proposer is whitelisted.

19.3 Parliament Contract

19.3.1 Actions

Initialize

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

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

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

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

CreateOrganization

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

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

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

Creates parliament organization with input data.

- **CreateOrganizationInput**
  - **ProposalReleaseThreshold**: the threshold for releasing the proposal.
  - **proposer authority required**: setting this to true can allow anyone to create proposals.
  - **parliament member proposing allowed**: setting this to true can allow parliament member to create proposals.
- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the value to be divided by `10000` for the minimum approval threshold in fraction.
  - **maximal rejection threshold**: the value to be divided by `10000` for the maximal rejection threshold in fraction.
  - **maximal abstention threshold**: the value to be divided by `10000` for the maximal abstention threshold in fraction.
  - **minimal vote threshold**: the value to be divided by `10000` for the minimal vote threshold in fraction.
- **Returns**
  - **Address**: the address of newly created organization.
- **Events**
  - **OrganizationCreated**
    * **organization address**: the address of newly created organization.

CreateOrganizationBySystemContract

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

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

message OrganizationCreated{
    option (aelf.is_event) = true;
```

(continues on next page)

(continued from previous page)

```
    aelf.Address organization_address = 1;
}
```

Creates parliament organization when called by system contract.

- **CreateOrganizationBySystemContractInput**
  - **CreateOrganizationInput**: the parameters of creating the organization.
  - **organization address feedback method**: organization address callback method which replies the organization address to caller contract.

note: *for CreateOrganizationInput see CreateOrganization*

- **Returns**
  - **Address**: the address of newly created organization.
- **Events**
  - **OrganizationCreated**
    - **organization address**: the address of newly created organization.

19.3.2 ACS3 specific methods

CreateProposal

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

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

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

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

- **CreateProposalInput**
  - **contract method name**: the name of the method to call after release.
  - **to address**: the address of the contract to call after release.
  - **expiration**: the timestamp at which this proposal will expire.
  - **organization address**: the address of the organization.
  - **proposal_description_url**: the url is used for proposal describing.

    - **token**: the token is for proposal id generation and with this token, proposal id can be calculated before proposing.

- **Returns**
    - **Hash**: id of the newly created proposal.
- **Events**
    - **ProposalCreated**
        - **proposal_id**: id of the created proposal.

Approve

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

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

This method is called to approve the specified proposal.

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

Reject

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

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

This method is called to reject the specified proposal.

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

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

Abstain

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

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

This method is called to abstain from the specified proposal.

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

Release

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

This method is called to release the specified proposal.

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

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

ChangeOrganizationThreshold

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

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

(continues on next page)

(continued from previous page)

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

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

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

- **ProposalReleaseThreshold**
  - **minimal approval threshold**: the new value for the minimum approval threshold.
  - **maximal rejection threshold**: the new value for the maximal rejection threshold.
  - **maximal abstention threshold**: the new value for the maximal abstention threshold.
  - **minimal vote threshold**: the new value for the minimal vote threshold.
- **Events**
  - **OrganizationThresholdChanged**
    * **organization_address**: the organization address.
    * **proposer_release_threshold**: the new release threshold.

ChangeOrganizationProposerWhiteList

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

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

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

This method overrides the list of whitelisted proposers.

- **ProposerWhiteList**:
  - **proposers**: the new value for the list.
- **Events**
  - **OrganizationWhiteListChanged**
    * **organization_address**: the organization address.
    * **proposer_white_list**: the new proposer whitelist.

CreateProposalBySystemContract

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

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

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

Used by system contracts to create proposals.

- **CreateProposalBySystemContractInput**
  - **CreateProposalInput**: the parameters of creating a proposal.
  - **origin proposer**: the actor that trigger the call.

note: *for CreateProposalInput see CreateProposal*

- **Returns**
  - **Address**: id of the newly created proposal.
- **Events**
  - **ProposalCreated**
    - **proposal_id**: id of the created proposal.

ClearProposal

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

Removes the specified proposal.

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

ValidateOrganizationExist

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

Checks the existence of an organization.

- **Address**: organization address to be checked.
- **Returns**
  - **BoolValue**: indicates whether the organization exists.

19.3.3 View methods

GetOrganization

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

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

Returns the organization with the provided organization address.

- **Address**: organization address.
- **Returns**
  - **Organization**
    - **proposer authority required**: indicates if proposals need authority to be created.
    - **organization_address**: organization address.
    - **organization hash**: organization id.
    - **ProposalReleaseThreshold**: the threshold.
    - **parliament member proposing allowed**: indicates if parliament member can propose to this organization.

note: *for ProposalReleaseThreshold see CreateOrganization*

GetDefaultOrganizationAddress

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

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

ValidateAddressIsParliamentMember

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

Validates if the provided address is a parliament member.

- **Address**: parliament member address to be checked.
- **Returns**
  - **BoolValue**: indicates whether provided address is one of parliament members.

GetProposerWhiteList

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

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

Returns a list of whitelisted proposers.

- **Returns**
  - **proposers**: the whitelisted proposers.

GetNotVotedPendingProposals

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

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

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

GetNotVotedProposals

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

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

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

CalculateOrganizationAddress

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

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

Calculates with input and returns the organization address.

- **CreateOrganizationInput**
  - **ProposalReleaseThreshold**: the threshold.
  - **proposer authority required**: setting this to true can allow anyone to create proposals.
  - **parliament member proposing allowed**: setting this to true can allow parliament member to create proposals.

note: *for ProposalReleaseThreshold see CreateOrganization*

- **Returns**
  - **Address**: organization address.

GetProposal

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

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

Get the proposal with the given id.

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

ValidateProposerInWhiteList

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

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

Checks if the proposer is whitelisted.

- **ValidateProposerInWhiteListInput**
  - **proposer**: the address to search/check.
  - **organization address**: address of the organization.
- **Returns**
  - **BoolValue**: indicates whether the proposer is whitelisted.

19.4 Consensus Contract

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

19.4.1 view methods

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

GetCurrentMinerList

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

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

Gets the list of current miners.

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

GetCurrentMinerPubkeyList

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

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

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

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

GetCurrentMinerListWithRoundNumber

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

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

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

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

- **Returns**
  - **miner list**: miners list.
  - **round number**: current round number.
- **MinerList**
  - **pubkeys**: miners' public keys.

GetRoundInformation

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

message SInt64Value
{
    sint64 value = 1;
}

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

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

(continues on next page)

(continued from previous page)

```
    aelf.Hash signature = 5;
    google.protobuf.Timestamp expected mining time = 6;
    sint64 produced blocks = 7;
    sint64 missed time slots = 8;
    string pubkey = 9;
    aelf.Hash previous in value = 12;
    sint32 supposed order of next round = 13;
    sint32 final order of next round = 14;
    repeated google.protobuf.Timestamp actual mining times = 15;
    map<string, bytes> encrypted pieces = 16;
    map<string, bytes> decrypted pieces = 17;
    sint32 produced tiny blocks = 18;
    sint64 implied irreversible block height = 19;
}
```

Gets information of the round specified as input.

- **SInt64Value**
  - **value**: round number.
- **Returns**
  - **round number**: round number.
  - **real time miners information**: public key => miner information.
  - **blockchain age**: current time minus block chain start time (if the round number is 1, the block chain age is 1), represented in seconds.
  - **extra block producer of previous round**: the public key (hexadecimal string) of the first miner, who comes from the last term, in the current term.
  - **term number**: the current term number.
  - **confirmed irreversible block height**: irreversible block height.
  - **confirmed irreversible block round number**: irreversible block round number.
  - **is miner list just changed**: is miner list different from the the miner list in the previous round.
  - **round id for validation**: round id, calculated by summing block producers' expecting time (second).
- **MinerInRound**
  - **order**: the order of miners producing block.
  - **is extra block producer**: The miner who is the first miner in the first round of each term.
  - **in value**: the previous miner's public key.
  - **out value**: the post miner's public key.
  - **signature**: self signature.
  - **expected mining time**: expected mining time.
  - **produced blocks**: produced blocks.
  - **missed time slots**: missed time slots.
  - **pubkey**: public key string.
  - **previous in value**: previous miner's public key.
  - **supposed order of next round**: evaluated order in next round.

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

GetCurrentRoundNumber

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

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

Gets the current round number.

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

GetCurrentRoundInformation

Gets the current round's information.

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

note: *for Round see GetRoundInformation*

GetPreviousRoundInformation

Gets the previous round information.

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

note: *for Round see GetRoundInformation*

GetCurrentTermNumber

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

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

Gets the current term number.

- **Returns**

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

GetCurrentWelfareReward

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

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

Gets the current welfare reward.

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

GetPreviousMinerList

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

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

Gets the miners in the previous term.

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

GetMinedBlocksOfPreviousTerm

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

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

Gets the number of mined blocks during the previous term.

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

GetNextMinerPubkey

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

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

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

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

GetCurrentMinerPubkey

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

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

Gets the current miner.

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

IsCurrentMiner

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

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

Query whether the miner is the current miner.

- **Address**
  - **value**: miner's address.
- **Returns**
  - **value**: indicates if the input miner is the current miner.

GetNextElectCountDown

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

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

Count down to the next election.

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

19.5 Election Contract

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

19.5.1 Actions

AnnounceElection

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

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

QuitElection

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

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

Vote

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

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

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

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

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

ChangeVotingOption

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

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

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

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

Withdraw

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

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

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

- **Hash**
  - **value**: transaction id.

SetVoteWeightProportion

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

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

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

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

19.5.2 View methods

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

GetCandidates

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

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

Gets all candidates' public keys.

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

GetVotedCandidates

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

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

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

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

GetCandidateInformation

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

message StringValue {
  string value = 1;
}

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

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

- **StringValue**
  - **value**: public key (hexadecimal string) of the candidate.
- **Returns**
  - **pubkey**: public key (represented by an hexadecimal string).

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

GetVictories

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

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

Gets the victories of the latest term.

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

GetTermSnapshot

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

message GetTermSnapshotInput {
    sint64 term_number = 1;
}

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

Gets the snapshot of the term provided as input.

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

GetMinersCount

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

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

Count miners.

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

GetElectionResult

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

message GetElectionResultInput {
    sint64 term_number = 1;
}

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

Gets an election result by term id.

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

GetElectorVote

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

message StringValue {
  string value = 1;
}

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

Gets the voter's information.

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

GetElectorVoteWithRecords

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

message StringValue {
  string value = 1;
}

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

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

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

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

  - **active voting record ids**: transaction ids, in which transactions you vote.
  - **withdrawn voting record ids**: transaction ids.
  - **active voted votes amount**: the number(excluding the withdrawn) of token you vote.
  - **all voted votes amount**: the number of token you have voted.
  - **active voting records**: records of the vote transaction with detail information.
  - **withdrawn votes records**: no records in this api.
  - **pubkey**: voter public key (byte string).
- **ElectionVotingRecord**
  - **voter**: voter address.
  - **candidate**: public key.
  - **amount**: vote amount.
  - **term number**: snapshot number.
  - **vote id**: transaction id.
  - **lock time**: time left to unlock token.
  - **unlock timestamp**: unlock date.
  - **withdraw timestamp**: withdraw date.
  - **vote timestamp**: vote date.
  - **is withdrawn**: indicates if the vote has been withdrawn.
  - **weight**: vote weight for sharing bonus.
  - **is change target**: whether vote others.

GetElectorVoteWithAllRecords

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

message StringValue {
  string value = 1;
}

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


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

(continues on next page)

(continued from previous page)

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

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

- **StringValue**
  - **value**: the public key (hexadecimal string) of voter.
- **Returns**
  - **active voting record ids**: transaction ids, in which transactions you vote.
  - **withdrawn voting record ids**: transaction ids.
  - **active voted votes amount**: the number(excluding the withdrawn) of token you vote.
  - **all voted votes amount**: the number of token you have voted.
  - **active voting records**: records of transactions that are active.
  - **withdrawn votes records**: records of transactions in which withdraw is true.
  - **pubkey**: voter public key (byte string).
- **ElectionVotingRecord**
  - **voter**: voter address.
  - **candidate**: public key.
  - **amount**: vote amount.
  - **term number**: snapshot number.
  - **vote id**: transaction id.
  - **lock time**: time left to unlock token.
  - **unlock timestamp**: unlock date.
  - **withdraw timestamp**: withdraw date.
  - **vote timestamp**: vote date.
  - **is withdrawn**: indicates if the vote has been withdrawn.
  - **weight**: vote weight for sharing bonus.
  - **is change target**: whether vote others.

GetCandidateVote

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

message StringValue {
  string value = 1;
}

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

Gets statistical information about vote transactions of a candidate.

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

GetCandidateVoteWithRecords

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

message StringValue {
  string value = 1;
}

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

(continues on next page)

(continued from previous page)

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

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

- **StringValue**
  - **value**: public key of the candidate.
- **Returns**
  - **obtained active voting record ids**: vote transaction ids.
  - **obtained withdrawn voting record ids**: withdraw transaction ids.
  - **obtained active voted votes amount**: the valid number of vote token in current.
  - **all obtained voted votes amount**: total number of vote token the candidate has got.
  - **obtained active voting records**: the records of the transaction without withdrawing.
  - **obtained withdrawn votes records**: no records in this api.
- **ElectionVotingRecord**
  - **voter** voter address.
  - **candidate** public key.
  - **amount** vote amount.
  - **term number** snapshot number.
  - **vote id** transaction id.
  - **lock time** time left to unlock token.
  - **unlock timestamp** unlock date.
  - **withdraw timestamp** withdraw date.
  - **vote timestamp** vote date.
  - **is withdrawn** indicates whether the vote has been withdrawn.
  - **weight** vote weight for sharing bonus.
  - **is change target** whether vote others.

GetCandidateVoteWithAllRecords

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

message StringValue {
```

(continues on next page)

(continued from previous page)

```
  string value = 1;
}

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

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

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

- **StringValue**
  - **value**: public key of the candidate.
- **Returns**
  - **obtained active voting record ids**: vote transaction ids.
  - **obtained withdrawn voting record ids**: withdrawn transaction ids.
  - **obtained active voted votes amount**: the valid number of vote token in current.
  - **all obtained voted votes amount**: total number of vote token the candidate has got.
  - **obtained active voting records**: the records of the transaction without withdrawing.
  - **obtained withdrawn votes records**: the records of the transaction withdrawing the vote token.
- **ElectionVotingRecord**
  - **voter**: voter address.
  - **candidate**: public key.
  - **amount**: vote amount.
  - **term number**: snapshot number.
  - **vote id**: transaction id.
  - **lock time**: time left to unlock token.
  - **unlock timestamp**: unlock date.

  - **withdraw timestamp**: withdraw date.
  - **vote timestamp**: vote date.
  - **is withdrawn**: indicates whether the vote has been withdrawn.
  - **weight**: vote weight for sharing bonus.
  - **is change target**: whether vote others.

GetVotersCount

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

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

Gets the total number of voters.

- **Returns**
  - **value**: number of voters.

GetVotesAmount

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

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

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

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

GetCurrentMiningReward

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

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

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

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

GetPageableCandidateInformation

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

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

message GetPageableCandidateInformationOutput {
    repeated CandidateDetail value = 1;
}

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

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

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

- **PageInformation**
  - **start**: start index.
  - **length**: number of records.
- **Returns**
  - **CandidateDetail**: candidates' detailed information.
- **CandidateDetail**
  - **candidate information**: candidate information.
  - **obtained votes amount**: obtained votes amount.
- **CandidateInformation**
  - **pubkey**: public key (hexadecimal string).
  - **terms**: indicate which terms have the candidate participated.
  - **produced blocks**: the number of blocks the candidate has produced.
  - **missed time slots**: the time the candidate failed to produce blocks.
  - **continual appointment count**: the time the candidate continue to participate in the election.
  - **announcement transaction id**: the transaction id that the candidate announce.
  - **is current candidate**: indicate whether the candidate can be elected in the current term.

GetMinerElectionVotingItemId

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

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

Gets the voting activity id.

- **Returns**
  - **value**: voting item id.

GetDataCenterRankingList

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

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

Gets the data center ranking list.

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

GetVoteWeightProportion

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

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

Gets VoteWeight Proportion.

note: *for VoteWeightProportion see SetVoteWeightProportion*

GetCalculateVoteWeight

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

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

Calculate the concrete vote weight according to your input.

- **VoteInformation**
  - **amount**: the vote amount.
  - **lock time**: the lock time your vote.
- **Returns**
  - **value**: vote weight calculated with your input and our function.

19.6 Genesis Contract

This page describes available methods on the Genesis Contract.

19.6.1 Actions

DeploySystemSmartContract

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

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

Deploys a system smart contract on chain.

- **SystemContractDeploymentInput**
  - **category**: contract type.
  - **code**: byte array of system contract code.
  - **name**: name hash of system contract.
  - **transaction_method_call_list**: list of methods called by system transaction.
- **Returns**
  - **value**: address of the deployed contract.

ProposeNewContract

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

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

Propose new contract deployment.

- **ContractDeploymentInput**
  - **category**: contract type (usually 0 for now)
  - **code**: byte array that represents the contract code
- **Returns**
  - **value:**: Hash of the **ContractDeploymentInput** object.

ProposeUpdateContract

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

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

Creates a proposal to update the specified contract.

- **ContractUpdateInput**
  - **address**: address of the contract to be updated
  - **code**: byte array of the contract's new code
- **Returns**
  - **value**: Hash of the **ContractUpdateInput** object.

ProposeContractCodeCheck

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

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

Propose to check the code of a contract.

- **ContractCodeCheckInput**
  - **contract_input**: byte array of the contract code to be checked
  - **is_contract_deployment**: whether the input contract is to be deployed or updated

  - **code_check_release_method**: method to call after code check complete (DeploySmartContract or UpdateSmartContract)
  - **proposed_contract_input_hash**: id of the proposed contract
  - **category**: contract category (always 0 for now)
- **Returns**
  - **value**: Hash of the proposed contract.

ReleaseApprovedContract

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

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

Releases a contract proposal which has been approved.

- **ReleaseContractInput**
  - **proposal_id**: hash of the proposal.
  - **proposed_contract_input_hash**: id of the proposed contract

ReleaseCodeCheckedContract

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

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

Release the proposal which has passed the code check.

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

DeploySmartContract

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

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

Deploys a smart contract on chain.

- **ContractDeploymentInput**

  - **category**: contract type (usually 0)
  - **code**: byte array of the contract code
- **Returns**
  - **value**: address of the deployed smart contract.

UpdateSmartContract

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

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

Updates a smart contract on chain.

- **ContractUpdateInput**
  - **address**: address of the smart contract to be updated
  - **code**: byte array of the updated contract code
- **Returns**
  - **value**: address of the updated smart contract.

Initialize

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

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

Initializes the genesis contract.

- **InitializeInput**
  - **contract_deployment_authority_required**: whether contract deployment/update requires authority.

ChangeGenesisOwner

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

Change the owner of the genesis contract.

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

SetContractProposerRequiredState

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

Set authority of contract deployment.

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

ChangeContractDeploymentController

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

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

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

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

ChangeCodeCheckController

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

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

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

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

SetInitialControllerAddress

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

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

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

GetContractDeploymentController

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

Returns **ContractDeploymentController** authority info.

- **Returns**: **ContractDeploymentController** authority info.

GetCodeCheckController

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

Returns **CodeCheckController** authority info.

- **Returns**: **CodeCheckController** authority info.

19.6.2 Views methods

CurrentContractSerialNumber

```
rpc CurrentContractSerialNumber (google.protobuf.Empty) returns (google.protobuf.
↪UInt64Value){}
```

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

- **Returns**
  - **value**: serial number of the genesis contract.

GetContractInfo

```
rpc GetContractInfo (aelf.Address) returns (ContractInfo){}

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

Gets detailed information about the specified contract.

- **Address**: address the contract

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

GetContractAuthor

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

Get author of the specified contract.

- **Address**: address of specified contract

- **Returns**
  - **value**: author of the specified contract.

GetContractHash

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

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

- **Address**: address of a contract
- **Returns**
  - **value**: the code hash of the contract.

GetContractAddressByName

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

Gets the address of a system contract by its name.

- **Hash**: name hash of the contract
- **Returns**
  - **value**: Address of the specified contract.

GetSmartContractRegistrationByAddress

```
rpc GetSmartContractRegistrationByAddress (aelf.Address) returns (aelf.
↪SmartContractRegistration){}

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

Gets the registration of a smart contract by its address.

- **Address** - address of a smart contract
- **Returns**
  - **value**: Registration object of the smart contract.

GetSmartContractRegistrationByCodeHash

```
rpc GetSmartContractRegistrationByCodeHash (aelf.Hash) returns (aelf.
↪SmartContractRegistration){}

message SmartContractRegistration {
```

(continues on next page)

(continued from previous page)

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

Gets the registration of a smart contract by code hash.

- **Hash** - contract code hash
- **Returns**
  - **value**: registration object of the smart contract.

ValidateSystemContractAddress

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

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

Validates whether the input system contract exists.

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

19.7 multi-token

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

19.7.1 Token life-cycle: creation, issuance and transfer

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

Create

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

message CreateInput {
    string symbol = 1;
    string token_name = 2;
    sint64 total_supply = 3;
    sint32 decimals = 4;
    aelf.Address issuer = 5;
```

(continues on next page)

(continued from previous page)

```
    bool is_burnable = 6;
    repeated aelf.Address lock_white_list = 7;
    bool is_profitable = 8;
    int32 issue_chain_id = 9;
}
```

The token contract permits the creation of an entirely new token and the first action needed before using a token is its creation.

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

Issue

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

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

Issuing some amount of tokens to an address is the action of increasing that addresses balance for the given token. The total amount of issued tokens must not exceed the total supply of the token and only the issuer (creator) of the token can issue tokens. Issuing tokens effectively increases the circulating supply.

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

Transfer

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

message TransferInput {
```

(continues on next page)

(continued from previous page)

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

Transferring tokens simply is the action of transferring a given amount of tokens from one address to another. The origin or source address is the signer of the transaction. The balance of the sender must be higher than the amount that is transferred.

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

TransferFrom

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

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

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

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

19.7.2 Allowances

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

Approve

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

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

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

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

UnApprove

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

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

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

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

19.7.3 Locking

Lock

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

message LockInput {
    aelf.Address address = 1;
    aelf.Hash lock_id = 2;
    string symbol = 3;
    string usage = 4;
    int64 amount = 5;
}
```

This method can be used to lock tokens.

- **LockInput**

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

Unlock

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

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

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

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

19.7.4 Burning tokens

Burn

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

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

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

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

19.7.5 Cross-chain

CrossChainCreateToken

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

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

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

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

CrossChainTransfer

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

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

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

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

CrossChainReceiveToken

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

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

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

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

SetSymbolsToPayTxSizeFee

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

message SymbolListToPayTxSizeFee{
    repeated SymbolToPayTxSizeFee symbols_to_pay_tx_size_fee = 1;
}

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

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

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

UpdateCoefficientsForContract

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

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

message CalculateFeePieceCoefficients {
    repeated sint32 value = 1;
}

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

This action sets methods used to calculate resource token fees.

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

UpdateCoefficientsForSender

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

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

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

This action sets methods used to calculate transaction fee.

note: *for CalculateFeeCoefficients see UpdateCoefficientsForContract*

AdvanceResourceToken

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

message AdvanceResourceTokenInput {
```

(continues on next page)

(continued from previous page)

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

This action transfers resource tokens to designated contract address.

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

TakeResourceTokenBack

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

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

This method takes token from contract address

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

ValidateTokenInfoExists

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

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

This method validates if the token exist.

- **ValidateTokenInfoExistsInput**
  - **symbol**: the token symbol.

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

TransferToContract

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

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

This method transfer token to token address.

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

InitializeAuthorizedController

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

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

ChangeUserFeeController

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

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

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

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

ChangeDeveloperController

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

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

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

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

19.7.6 View methods

GetTokenInfo

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

message GetTokenInfoInput {
    string symbol = 1;
}

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

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

- **GetTokenInfoInput**
  - **symbol**: the token for which you want the information.
- **Returns**
  - **symbol**: the symbol of the token.
  - **token_name**: the full name of the token.
  - **supply**: the current supply of the token.
  - **total_supply**: the total supply of the token.
  - **decimals**: the amount of decimal places this token has.
  - **issuer**: the address that created the token.
  - **is_burnable**: a flag indicating if this token is burnable.

   - **is_profitable**: a flag indicating if this token is profitable.
   - **issue_chain_id**: the chain of this token.
   - **burned**: the amount of burned tokens.

GetNativeTokenInfo

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

note: *for TokenInfo see GetTokenInfo*

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

GetResourceTokenInfo

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

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

note: *for TokenInfo see GetTokenInfo*

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

GetBalance

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

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

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

This view method returns the balance of an address.

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

GetAllowance

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

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

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

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

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

IsInWhiteList

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

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

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

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

GetLockedAmount

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

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

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

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

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

GetCrossChainTransferTokenContractAddress

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

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

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

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

GetPrimaryTokenSymbol

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

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

GetCalculateFeeCoefficientOfContract

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

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

message CalculateFeePieceCoefficients {
    repeated sint32 value = 1;
}

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

This view method returns the resource tokens fee calculation method.

- **CalculateFeeCoefficients**: resource fee type.

Output note: *for CalculateFeeCoefficients see UpdateCoefficientsForContract*

GetCalculateFeeCoefficientOfSender

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

This view method returns transaction fee's calculation method.

note: *for CalculateFeeCoefficients see GetCalculateFeeCoefficientForContract*

GetSymbolsToPayTxSizeFee

```
rpc GetSymbolsToPayTxSizeFee (google.protobuf.Empty) returns␣
↪(SymbolListToPayTxSizeFee){}

message SymbolListToPayTxSizeFee{
    repeated SymbolToPayTxSizeFee symbols_to_pay_tx_size_fee = 1;
}

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

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

note: *for SymbolListToPayTxSizeFee see SetSymbolsToPayTxSizeFee*

GetDeveloperFeeController

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

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

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

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

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

GetUserFeeController

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

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

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

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

- **UserFeeController**
  - **root_controller** root controller information.
  - **parliament_controller** parliament controller information.
  - **referendum_controller** referndum controller information.
- **Returns**

  - **contract_address**: in which contract the organization is created.
  - **owner_address/**: organization address

GetSideChainRentalControllerCreateInfo

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

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

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

GetResourceUsage

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

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

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

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

GetOwningRental

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

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

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

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

GetOwningRentalUnitValue

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

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

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

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

19.8 Profit Contract

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

19.8.1 Actions

Scheme Creation

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

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

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

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

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

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

Add sub-scheme

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

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

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

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

Remove sub-scheme

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

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

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

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

Add beneficiary

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

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

message BeneficiaryShare {
```

(continues on next page)

(continued from previous page)

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

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

- **AddBeneficiaryInput**
  - **scheme id**: scheme id.
  - **beneficiary share**: share information to beneficiary.
  - **end period**: end time.
- **BeneficiaryShare**
  - **beneficiary**: beneficiary address
  - **shares**: shares attributed to this beneficiary.

Remove beneficiary

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

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

Removes a beneficiary from a scheme.

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

Add beneficiaries

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

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

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

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

- **AddBeneficiariesInput**
  - **scheme id**: scheme id.

  - **beneficiary shares**: share information to beneficiaries.
  - **end period**: end time.
- **BeneficiaryShare**
  - **beneficiary**: beneficiary address.
  - **shares**: shares to beneficiary.

Remove beneficiaries

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

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

Remove beneficiaries from a scheme.

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

Profit contribution

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

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

Contribute profit to a scheme.

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

Claim profits

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

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

(continues on next page)

(continued from previous page)

```
    aelf.Address beneficiary = 3;
}
```

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

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

Distribute profits

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

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

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

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

Reset manager

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

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

Reset the manager of a scheme.

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

19.8.2 View methods

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

GetManagingSchemeIds

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

message GetManagingSchemeIdsInput {
    aelf.Address manager = 1;
}

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

Get all schemes created by the specified manager.

- **GetManagingSchemeIdsInput**
  - **manager**: manager's address.
- **Returns**
  - **scheme ids**: list of scheme ids.

GetScheme

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

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

- **Hash**
  - **value**: scheme id.
- **SchemeBeneficiaryShare**
  - **scheme id**: sub scheme's id.
  - **shares**: sub scheme shares.

GetSchemeAddress

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

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

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

- **SchemePeriod**
  - **scheme id**: scheme id.
  - **period**: period number.
- **Returns**
  - **value**: scheme's virtual address.

GetDistributedProfitsInfo

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

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

message DistributedProfitsInfo {
    sint64 total_shares = 1;
    map<string, sint64> profits_amount = 2;
    bool is_released = 3;
}
```

Get distributed profits Info for a given period.

- **SchemePeriod**
  - **scheme_id**: scheme id.
  - **period**: term number.
- **Returns**
  - **total shares**: total shares, -1 indicates failed to get the information.
  - **profits amount**: token symbol => reside amount.
  - **is released**: is released.

GetProfitDetails

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

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

message ProfitDetails {
    repeated ProfitDetail details = 1;
}

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

Gets a beneficiaries profit details for a given scheme.

- **GetProfitDetailsInput**
  - **scheme id** scheme id.
  - **beneficiary** beneficiary.
- **Returns**

    - **details** profit details.

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

GetProfitAmount

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

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

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

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

- **ClaimProfitsInput**
    - **scheme_id**: scheme id.
    - **symbol**: token symbol.
- **Returns**
    - **value**: amount of tokens.

19.9 Cross Chain Contract

Defines C# API functions for cross chain contract.

19.9.1 Actions

ProposeCrossChainIndexing

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

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

message SideChainBlockData {
```

(continues on next page)

(continued from previous page)

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

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

message CrossChainExtraData {
    aelf.Hash transaction_status_merkle_tree_root = 1;
}

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

Propose once cross chain indexing.

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

ReleaseCrossChainIndexing

```
rpc ReleaseCrossChainIndexing(ReleaseCrossChainIndexingProposalInput) returns (google.
↪protobuf.Empty) {

message ReleaseCrossChainIndexingProposalInput{
    repeated int32 chain_id_list = 1;
}
```

Release cross chain indexing proposal.

- **ReleaseCrossChainIndexingProposalInput**
  - **chain_id_list**: chain id list to release

Initialize

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

message InitializeInput
{
    int32 parent_chain_id = 1;
    int64 creation_height_on_parent_chain = 2;
    bool is_privilege_preserved = 3;
}
```

Initialize cross-chain-contract.

- **InitializeInput**
  - **parent_chain_id**: id of parent chain
  - **creation_height_on_parent_chain**: height of side chain creation on parent chain
  - **is_privilege_preserved**: true if chain privilege needed, otherwise false

RequestSideChainCreation

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

message SideChainCreationRequest {
    int64 indexing_price = 1;
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    SideChainTokenCreationRequest side_chain_token_creation_request = 4;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 5;
    map<string, int32> initial_resource_amount = 6;
}

message SideChainTokenCreationRequest{
    string side_chain_token_symbol = 1;
    string side_chain_token_name = 2;
    int64 side_chain_token_total_supply = 3;
    int32 side_chain_token_decimals = 4;
    bool is_side_chain_token_burnable = 5;
```

(continues on next page)

(continued from previous page)

```
}

message SideChainTokenInitialIssue{
    aelf.Address address = 1;
    int64 amount = 2;
}
```

Request side chain creation.

- **SideChainCreationRequest**
  - **indexing_price**: indexing fee.
  - **locked_token_amount**: initial locked balance for a new side chain.
  - **is_privilege_preserved**: creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.
  - **side_chain_token_initial_issue_list**: a list of accounts and amounts that will be issued when the chain starts.
  - **initial_resource_amount**: the initial rent resources.
- **SideChainTokenCreationRequest**
  - **side_chain_token_symbol**: side chain token symbol.
  - **side_chain_token_name**: side chain token name.
  - **side_chain_token_total_supply**: total supply of side chain token.
  - **side_chain_token_decimals**: side chain token decimal.

ReleaseSideChainCreation

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

message ReleaseSideChainCreationInput {
    aelf.Hash proposal_id = 1;
}
```

Release side chain creation proposal and side chain will be created.

- **ReleaseSideChainCreationInput**
  - **proposal_id**: side chain creation proposal id

CreateSideChain

```
rpc CreateSideChain (CreateSideChainInput) returns (google.protobuf.Int32Value) {}

message CreateSideChainInput{
    SideChainCreationRequest side_chain_creation_request = 1;
    aelf.Address proposer = 2;
}

message SideChainCreationRequest {
```

(continues on next page)

(continued from previous page)

```
    int64 indexing_price = 1;
    int64 locked_token_amount = 2;
    bool is_privilege_preserved = 3;
    string side_chain_token_symbol = 4;
    string side_chain_token_name = 5;
    int64 side_chain_token_total_supply = 6;
    int32 side_chain_token_decimals = 7;
    bool is_side_chain_token_burnable = 8;
    bool is_side_chain_token_profitable = 9;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 10;
    map<string, int32> initial_resource_amount = 11;
}

message SideChainTokenInitialIssue {
    aelf.Address address = 1;
    int64 amount = 2;
}
```

Create a new side chain, this is be triggered by an organization address.

- **CreateSideChainInput**
  - **proposer**: the proposer of the proposal that triggered this method.
  - **SideChainCreationRequest**: the parameters of creating side chain.

note: *for SideChainCreationRequest see RequestSideChainCreation*

- **Returns** Id of a new side chain

SetInitialSideChainLifetimeControllerAddress

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

Sets the initial **SideChainLifetimeController** address which should be parliament organization by default.

- **Address** : the owner's address.

SetInitialIndexingControllerAddress

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

Sets the initial **CrossChainIndexingController** address which should be parliament organization by default.

- **Address**: the owner's address.

ChangeCrossChainIndexingController

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

message AuthorityInfo {
```

(continues on next page)

(continued from previous page)

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

Changes the cross chain indexing controller.

- **AuthorityInfo**
  - **contract_address**: the address of the contract that generated the controller.
  - **owner_address**: the address of the controller.

ChangeSideChainLifetimeController

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

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

Changes the side chain's lifetime controller.

note: *for AuthorityInfo see ChangeCrossChainIndexingController*

ChangeSideChainIndexingFeeController

```
rpc ChangeSideChainIndexingFeeController(ChangeSideChainIndexingFeeControllerInput)␣
↪returns (google.protobuf.Empty){}

message ChangeSideChainIndexingFeeControllerInput{
    int32 chain_id = 1;
    AuthorityInfo authority_info = 2;
}

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

Changes indexing fee adjustment controller for specific side chain.

- **ChangeSideChainIndexingFeeControllerInput**
  - **chain_id**: side chain id.
  - **authority_info**
    - **contract_address**: the address of the contract that generated the controller.
    - **owner_address**: the address of the controller.

Recharge

```
rpc Recharge (RechargeInput) returns (google.protobuf.Empty) {}
message RechargeInput {
    int32 chain_id = 1;
    int64 amount = 2;
}
```

Recharge for specified side chain.

- **RechargeInput**
  - **chain_id**: id of the side chain
  - **amount**: the token amount to recharge

RecordCrossChainData

```
rpc RecordCrossChainData (RecordCrossChainDataInput) returns (google.protobuf.Empty){}

message RecordCrossChainDataInput{
    CrossChainBlockData proposed_cross_chain_data = 1;
    aelf.Address proposer = 2;
}

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

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

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

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

Index block data of parent chain and side chain. Only **CrossChainIndexingController** is permitted to invoke this method.

- **RecordCrossChainDataInput**
  - **CrossChainBlockData**: cross chain block data.
  - **proposer**: the proposal's address.

note: *for CrossChainBlockData see ProposeCrossChainIndexing*

AdjustIndexingFeePrice

```
rpc AdjustIndexingFeePrice(AdjustIndexingFeeInput)returns(google.protobuf.Empty){}

message AdjustIndexingFeeInput{
    int32 side_chain_id = 1;
    int64 indexing_fee = 2;
}
```

Adjust side chain indexing fee. Only **IndexingFeeController** is permitted to invoke this method.

- **AdjustIndexingFeeInput**
- **side_chain_id**: side chain id
- **indexing_fee**: indexing fee to be set

DisposeSideChain

```
rpc DisposeSideChain (google.protobuf.Int32Value) returns (google.protobuf.Int32Value)
↪{}
```

Dispose the specified side chain. Only **SideChainLifetimeController** is permitted to invoke this method.

- **Int32Value**: the id of side chain to be disposed
- **Returns**
  - **value**: the id of disposed chain

19.9.2 View methods

Get pending cross chain indexing proposal info.

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

note: *for CrossChainBlockData see ProposeCrossChainIndexing*

GetCrossChainIndexingController

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

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

Get indexing fee adjustment controller for specific side chain.

note: *for AuthorityInfo see ChangeCrossChainIndexingController*

GetSideChainLifetimeController

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

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

Get the side chain's lifetime controller.

note: *for AuthorityInfo see ChangeCrossChainIndexingController*.

GetSideChainIndexingFeeController

```
rpc GetSideChainIndexingFeeController(google.protobuf.Int32Value) returns␣
↪(AuthorityInfo){}

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

Get side chain indexing fee.

- **Int32Value**: side chain id

note: *for AuthorityInfo see ChangeCrossChainIndexingController*

GetSideChainIndexingFeePrice

```
rpc GetSideChainIndexingFeePrice(google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value){}
```

Get side chain indexing fee.

- **Int32Value**: side chain id
- **Returns**
  - **value**: indexing fee price

VerifyTransaction

```
rpc VerifyTransaction (VerifyTransactionInput) returns (google.protobuf.BoolValue){}

message VerifyTransactionInput {
    aelf.Hash transaction_id = 1;
    aelf.MerklePath path = 2;
    int64 parent_chain_height = 3;
```

(continues on next page)

(continued from previous page)

```
    int32 verified_chain_id = 4;
}
```

Transaction cross chain verification.

- **VerifyTransactionInput**
  - **transaction_id**: transaction id
  - **path**: merkle path for the transaction
  - **parent_chain_height**: height of parent chain indexing this transaction
  - **verified_chain_id**: id of the chain to be verified
- **Returns**
  - **value**: true if verification succeeded, otherwise false.

LockedAddress

```
rpc GetSideChainCreator (google.protobuf.Int32Value) returns (aelf.Address){}
```

Get side chain creator address.

- **Int32Value**: id of side chain
- **Returns**
  - **value**: address of side chain creator.

GetChainStatus

```
rpc GetChainStatus (google.protobuf.Int32Value) returns (GetChainStatusOutput){}

message GetChainStatusOutput{
    SideChainStatus status = 1;
}

enum SideChainStatus
{
    FATAL = 0;
    ACTIVE = 1;
    INDEXING_FEE_DEBT = 2;
    TERMINATED = 3;
}
```

Gets the current status of the specified side chain.

- **Int32Value**: id of side chain.
- **Returns** Current status of side chain.
  - **fatal**: currently no meaning.
  - **active**: the side-chain is being indexed.
  - **insufficient fee debt**: debt for indexing fee to be payed off.
  - **terminated**: the side chain cannot be indexed anymore.

GetSideChainHeight

```
rpc GetSideChainHeight (google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value){}
```

Get current height of the specified side chain.

- **Int32Value**: id of side chain
- **Returns**
  - **value**: current height of the side chain.

GetParentChainHeight

```
rpc GetParentChainHeight (google.protobuf.Empty) returns (google.protobuf.Int64Value)
↪{}
```

Get recorded height of parent chain

- **Returns**
  - **value**: height of parent chain.

GetParentChainId

```
rpc GetParentChainId (google.protobuf.Empty) returns (google.protobuf.Int32Value){}
```

Get id of the parent chain. This interface is only for side chain.

- **Returns**
  - **value**: parent chain id.

GetSideChainBalance

```
rpc GetSideChainBalance (google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value){}
```

Get the balance for side chain indexing.

- **Int32Value**: id of side chain
- **Returns**
  - **value**: balance for side chain indexing.

GetSideChainIndexingFeeDebt

```
rpc GetSideChainIndexingFeeDebt (google.protobuf.Int32Value) returns (google.protobuf.
↪Int64Value){}
```

Get indexing debt for side chain.

- **Int32Value**: id of side chain

- **Returns**
  - **value**: side chain indexing debt. Returns zero if no debt.

GetSideChainIdAndHeight

```
rpc GetSideChainIdAndHeight (google.protobuf.Empty) returns (ChainIdAndHeightDict){}

message ChainIdAndHeightDict
{
    map<int32, int64> id_height_dict = 1;
}
```

Get id and recorded height of side chains.

- **Returns**
  - **ChainIdAndHeightDict**: A map contains id and height of side chains

GetSideChainIndexingInformationList

```
rpc GetSideChainIndexingInformationList (google.protobuf.Empty) returns␣
↪(SideChainIndexingInformationList){}

message SideChainIndexingInformationList
{
    repeated SideChainIndexingInformation indexing_information_list = 1;
}

message SideChainIndexingInformation
{
    int32 chain_id = 1;
    int64 indexed_height = 2;
}
```

Get indexing information of side chains.

- **Returns**
  - **SideChainIndexingInformationList**: A list contains indexing information of side chains

GetAllChainsIdAndHeight

```
rpc GetAllChainsIdAndHeight (google.protobuf.Empty) returns (ChainIdAndHeightDict){}

message ChainIdAndHeightDict
{
    map<int32, int64> id_height_dict = 1;
}
```

Get id and recorded height of all chains.

- **Returns -ChainIdAndHeightDict**: A map contains id and height of all chains

GetIndexedSideChainBlockDataByHeight

```
rpc GetIndexedSideChainBlockDataByHeight (google.protobuf.Int64Value) returns␣
↪(IndexedSideChainBlockData){}

message IndexedSideChainBlockData
{
    repeated acs7.SideChainBlockData side_chain_block_data = 1;
}
message SideChainBlockData
{
    int64 height = 1;
    aelf.Hash block_header_hash = 2;
    aelf.Hash transaction_merkle_tree_root = 3;
    int32 chain_id = 4;
}
```

Get block data of indexed side chain by height

- **Int64Value**: height of side chain
- **Returns**
  - **side_chain_block_data**: block data of side chain.

GetBoundParentChainHeightAndMerklePathByHeight

```
rpc GetBoundParentChainHeightAndMerklePathByHeight (google.protobuf.Int64Value)␣
↪returns (CrossChainMerkleProofContext){}

message CrossChainMerkleProofContext
{
    int64 bound_parent_chain_height = 1;
    aelf.MerklePath merkle_path_for_parent_chain_root = 2;
}
message MerklePath
{
    repeated MerklePathNode merklePathNodes = 1;
}
message MerklePathNode
{
    Hash hash = 1;
    bool isLeftChildNode = 2;
}
```

Get merkle path bound up with side chain

- **Int64Value**: height of side chain.
- **Returns**
  - **bound_parent_chain_height**: height of parent chain bound up with side chain.
  - **merkle_path_from_parent_chain**: merkle path generated from parent chain.

GetChainInitializationData

```
rpc GetChainInitializationData (google.protobuf.Int32Value) returns␣
↪(ChainInitializationData){}

message ChainInitializationData
{
    int32 chain_id = 1;
    aelf.Address creator = 2;
    google.protobuf.Timestamp creation_timestamp = 3;
    int64 creation_height_on_parent_chain = 4;
    bool chain_creator_privilege_preserved = 5;
    aelf.Address parent_chain_token_contract_address = 6;
    ChainInitializationConsensusInfo chain_initialization_consensus_info = 7;
    bytes native_token_info_data = 8;
    ResourceTokenInfo resource_token_info = 9;
    ChainPrimaryTokenInfo chain_primary_token_info = 10;
}

message ChainInitializationConsensusInfo{
    bytes initial_miner_list_data = 1;
}

message ResourceTokenInfo{
    bytes resource_token_list_data = 1;
    map<string, int32> initial_resource_amount = 2;
}

message ChainPrimaryTokenInfo{
    bytes chain_primary_token_data = 1;
    repeated SideChainTokenInitialIssue side_chain_token_initial_issue_list = 2;
}

message SideChainTokenInitialIssue{
    aelf.Address address = 1;
    int64 amount = 2;
}
```

Get initialization data for specified side chain.

- **Int32Value**: id of side chain.
- **Returns**
  - **chain_id**: id of side chain.
  - **creator**: side chain creator.
  - **creation_timestamp**: timestamp for side chain creation.
  - **creation_height_on_parent_chain**: height of side chain creation on parent chain.
  - **chain_creator_privilege_preserved**: creator privilege boolean flag: True if chain creator privilege preserved, otherwise false.
  - **parent_chain_token_contract_address**: parent chain token contract address.
  - **chain_initialization_consensus_info**: initial miner list information.
  - **native_token_info_data**: native token info.
  - **resource_token_info**: resource token information.

  - **chain_primary_token_info**: chain priamry token information.
- **ChainInitializationConsensusInfo**
  - **initial_consensus_data**: initial consensus data.
- **ChainInitializationConsensusInfo**
  - **resource_token_list_data**: resource token list data.
  - **initial_resource_amount**: initial resource token amount.
- **ChainPrimaryTokenInfo**
  - **chain_primary_token_data**: side chain primary token data.
  - **side_chain_token_initial_issue_list**: side chain primary token initial issue list.

19.10 Treasury Contract

The Treasury contract is essentially used for distributing bonus' to voters and candidates during the election process.

19.10.1 Actions

Donate

```
rpc Donate (DonateInput) returns (google.protobuf.Empty){}

message DonateInput {
    string symbol = 1;
    sint64 amount = 2;
}

message DonationReceived {
    aelf.Address from = 1 [(aelf.is_indexed) = true];
    aelf.Address to = 2 [(aelf.is_indexed) = true];
    string symbol = 3 [(aelf.is_indexed) = true];
    sint64 amount = 4 [(aelf.is_indexed) = true];
    string memo = 5;
}
```

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.

- **DonateInput**
  - **symbol**: token symbol.
  - **amount**: token amount.
- **Event**
  - **DonationReceived**
    - **from**: from address.
    - **to**: to address.
    - **symbol**: token symbol.
    - **amount**: amount of token.

    * **memo**: memo.

Donate all tokens

```
rpc DonateAll (DonateAllInput) returns (google.protobuf.Empty){}

message DonateAllInput {
    string symbol = 1;
}
```

Donate all token (transfer to native token) from caller to the treasury (by calling **Donate** described above).

- **DonateAllInput**
  - **symbol**: token symbol.

SetDistributingSymbolList

```
rpc SetDistributingSymbolList (SymbolList) returns (google.protobuf.Empty){}

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

Set a token list that can be used to distribute.

- **SymbolList**
  - **value**: token symbol list.

SetDividendPoolWeightSetting

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

message DividendPoolWeightSetting {
    int32 citizen_welfare_weight = 1;
    int32 backup_subsidy_weight = 2;
    int32 miner_reward_weight = 3;
}
```

Set weight for the three activities.

- **DividendPoolWeightSetting**
  - **citizen welfare weight**: citizen welfare weight.
  - **backup subsidy weight**: backup subsidy weight.
  - **miner reward weight**: miner reward weight.

SetMinerRewardWeightSetting

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

message MinerRewardWeightSetting {
    int32 basic_miner_reward_weight = 1;
    int32 votes_weight_reward_weight = 2;
    int32 re_election_reward_weight = 3;
}
```

Set weight for the three activities composing of miner reward activity.

- **MinerRewardWeightSetting**
  - **basic miner reward weight**: basic miner reward weight.
  - **votes weight reward weight**: votes weight reward weight.
  - **re-election reward weight**: re-election reward weight.

ChangeTreasuryController

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

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

Change the controller who is able to update symbol list and activities' weight above.

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

19.10.2 View methods

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

GetCurrentTreasuryBalance

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

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

Get the Treasury's total balance of the native token from the Treasury.

- **Returns**
  - **value**: amount of native token.

GetWelfareRewardAmountSample

```
rpc GetWelfareRewardAmountSample (GetWelfareRewardAmountSampleInput) returns␣
↪(GetWelfareRewardAmountSampleOutput){}

message GetWelfareRewardAmountSampleInput {
    repeated sint64 value = 1;
}

message GetWelfareRewardAmountSampleOutput {
    repeated sint64 value = 1;
}
```

Test the welfare bonus gotten base on 10000 Vote Token. The input is a array of locking time, and the output is the corresponding welfare.

- **GetWelfareRewardAmountSampleInput**
  - **value**: a array of locking time.
- **Returns**
  - **value**: a array of welfare.

GetTreasurySchemeId

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

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

Get treasury scheme id. If it does not exist, it will return hash.empty.

- **Returns**
  - **value**: scheme id.

GetDistributingSymbolList

```
rpc GetDistributingSymbolList (google.protobuf.Empty) returns (SymbolList){}

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

Get the symbol list that can be used to distribute.

note: *for SymbolList see SetDistributingSymbolList*

GetDividendPoolWeightProportion

```
rpc GetDividendPoolWeightProportion (google.protobuf.Empty) returns␣
↪(DividendPoolWeightProportion){}

message DividendPoolWeightProportion {
    SchemeProportionInfo citizen_welfare_proportion_info = 1;
    SchemeProportionInfo backup_subsidy_proportion_info = 2;
    SchemeProportionInfo miner_reward_proportion_info = 3;
}

message SchemeProportionInfo{
    aelf.Hash scheme_id = 1;
    int32 proportion = 2;
}
```

Get activities's weight expressed as a percentage

- **Returns**
  - **citizen welfare proportion info**: citizen welfare proportion info.
  - **backup subsidy proportion info**: backup subsidy proportion info.
  - **miner reward proportion info**: miner reward proportion info.
- **SchemeProportionInfo**
  - **scheme id**: scheme id
  - **proportion**: the weight expressed as a percentage.

GetMinerRewardWeightProportion

```
rpc GetMinerRewardWeightProportion (google.protobuf.Empty) returns␣
↪(MinerRewardWeightProportion){}

message MinerRewardWeightProportion {
    SchemeProportionInfo basic_miner_reward_proportion_info = 1;
    SchemeProportionInfo votes_weight_reward_proportion_info = 2;
    SchemeProportionInfo re_election_reward_proportion_info = 3;
}
```

Get the weight expressed as a percentage of the activities composing of miner reward.

note: *for MinerRewardWeightProportion see GetDividendPoolWeightProportion*

GetTreasuryController

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

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

Get this contract's controller.

note: *for AuthorityInfo see ChangeTreasuryController*

19.11 Vote Contract

The Vote contract is an abstract layer for voting. Developers implement concrete voting activities by calling this contract.

19.11.1 Actions

Voting for Block Producers

```
rpc Register (VotingRegisterInput) returns (google.protobuf.Empty){}

message VotingRegisterInput {
    google.protobuf.Timestamp start_timestamp = 1;
    google.protobuf.Timestamp end_timestamp = 2;
    string accepted_currency = 3;
    bool is_lock_token = 4;
    sint64 total_snapshot_number = 5;
    repeated string options = 6;
}
```

To build a voting activity, the developer should register first.

- **VotingRegisterInput**
  - **start timestamp**: activity start time.
  - **end timestamp**: activity end time.
  - **accepted currency**: the token symbol which will be accepted.
  - **is lock token**: indicates whether the token will be locked after voting.
  - **total snapshot number**: number of terms.
  - **options**: default candidate.

Vote

```
rpc Vote (VoteInput) returns (google.protobuf.Empty){}

message VoteInput {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 amount = 3;
    string option = 4;
    aelf.Hash vote_id = 5;
    bool is_change_target = 6;
}

message Voted {
    option (aelf.is_event) = true;
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp vote_timestamp = 5;
```

(continues on next page)

(continued from previous page)

```
    string option = 6;
    aelf.Hash vote_id = 7;
}
```

After building successfully a voting activity, others are able to vote.

- **VoteInput**
  - **voting item id**: indicates which voting activity the user participate in.
  - **voter**: voter's address.
  - **amount**: vote amount.
  - **option**: candidate's public key.
  - **vote id**: transaction id.
  - **is change target**: indicates whether the option is changed.
- **Event**
  - **Voted**
    * **voting item id**: voting activity id.
    * **voter**: voter's address.
    * **snapshot number**: the current round.
    * **amount**: vote amount.
    * **vote timestamp**: vote time.
    * **option**: the candidate's public key.
    * **vote id**: transaction id.

Withdraw

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

message WithdrawInput {
    aelf.Hash vote_id = 1;
}

message Withdrawn {
    aelf.Hash vote_id = 1;
}
```

A voter can withdraw the token after the lock time.

- **WithdrawInput**
  - **vote id**: transaction id.
- **Event**
  - **Withdrawn**
    * **vote id**: transaction id.

TakeSnapshot

```
rpc TakeSnapshot (TakeSnapshotInput) returns (google.protobuf.Empty){}

message TakeSnapshotInput {
    aelf.Hash voting_item_id = 1;
    sint64 snapshot_number = 2;
}
```

Distributes profits and saves the state every round.

- **TakeSnapshotInput**
  - **voting item id**: voting activity id.
  - **snapshot number**: the round number.

AddOption

```
rpc AddOption (AddOptionInput) returns (google.protobuf.Empty){
}

message AddOptionInput {
    aelf.Hash voting_item_id = 1;
    string option = 2;
}
```

Adds an option (a choice) to a voting activity.

- **AddOptionInput**
  - **voting item id**: vote activity id.
  - **option**: the new option.

AddOptions

```
rpc AddOptions (AddOptionsInput) returns (google.protobuf.Empty){
}

message AddOptionsInput {
    aelf.Hash voting_item_id = 1;
    repeated string options = 2;
}
```

Adds multiple options (choices) to a voting activity.

- **AddOptionsInput**
  - **voting item id**: voting activity id.
  - **option**: the list of new options.

RemoveOption

```
rpc RemoveOption (RemoveOptionInput) returns (google.protobuf.Empty){
}

message RemoveOptionInput {
    aelf.Hash voting_item_id = 1;
    string option = 2;
}
```

Removes an option from a voting activity.

- **RemoveOptionInput**
  - **voting item id**: voting activity id.
  - **option**: the option to remove.

RemoveOptions

```
rpc RemoveOptions (RemoveOptionsInput) returns (google.protobuf.Empty){}

message RemoveOptionsInput {
    aelf.Hash voting_item_id = 1;
    repeated string options = 2;
}
```

Removes multiple options from a voting activity.

- **RemoveOptionsInput**
  - **voting item id**: voting activity id.
  - **option**: the options to remove.

19.11.2 View methods

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

GetVotingItem

```
rpc GetVotingItem (GetVotingItemInput) returns (VotingItem){
}

message GetVotingItemInput {
    aelf.Hash voting_item_id = 1;
}

message VotingItem {
    aelf.Hash voting_item_id = 1;
    string accepted_currency = 2;
    bool is_lock_token = 3;
    sint64 current_snapshot_number = 4;
    sint64 total_snapshot_number = 5;
    repeated string options = 6;
    google.protobuf.Timestamp register_timestamp = 7;
    google.protobuf.Timestamp start_timestamp = 8;
```

(continues on next page)

(continued from previous page)

```
    google.protobuf.Timestamp end_timestamp = 9;
    google.protobuf.Timestamp current_snapshot_start_timestamp = 10;
    aelf.Address sponsor = 11;
}
```

Gets the information related to a voting activity.

- **GetVotingItemInput**
  - **voting item id**: voting activity id.
- **Returns**
  - **voting item id**: voting activity id.
  - **accepted currency**: vote token.
  - **is lock token**: indicates if the token will be locked after voting.
  - **current snapshot number**: current round.
  - **total snapshot number**: total number of round.
  - **register timestamp**: register time.
  - **start timestamp**: start time.
  - **end timestamp**: end time.
  - **current snapshot start timestamp**: current round start time.
  - **sponsor**: activity creator.

GetVotingResult

```
rpc GetVotingResult (GetVotingResultInput) returns (VotingResult){}

message GetVotingResultInput {
    aelf.Hash voting_item_id = 1;
    sint64 snapshot_number = 2;
}

message VotingResult {
    aelf.Hash voting_item_id = 1;
    map<string, sint64> results = 2; // option -> amount
    sint64 snapshot_number = 3;
    sint64 voters_count = 4;
    google.protobuf.Timestamp snapshot_start_timestamp = 5;
    google.protobuf.Timestamp snapshot_end_timestamp = 6;
    sint64 votes_amount = 7;
}
```

Gets a voting result according to the provided voting activity id and round number.

- **GetVotingResultInput**
  - **voting item id**: voting activity id.
  - **snapshot number**: round number.
- **Returns**:

  - **voting item id**: voting activity id.
  - **results**: candidate => vote amount.
  - **snapshot number**: round number.
  - **voters count**: how many voters.
  - **snapshot start timestamp**: start time.
  - **snapshot end timestamp**: end time.
  - **votes amount** total votes(excluding withdraws).

GetLatestVotingResult

```
rpc GetLatestVotingResult (aelf.Hash) returns (VotingResult){}

message Hash
{
    bytes value = 1;
}

message VotingResult {
    aelf.Hash voting_item_id = 1;
    map<string, sint64> results = 2; // option -> amount
    sint64 snapshot_number = 3;
    sint64 voters_count = 4;
    google.protobuf.Timestamp snapshot_start_timestamp = 5;
    google.protobuf.Timestamp snapshot_end_timestamp = 6;
    sint64 votes_amount = 7;
}
```

Gets the latest result of the provided voting activity.

- **Hash**
  - **value**: voting activity id.
- **Returns**
  - **voting item id**: voting activity id.
  - **results**: candidate => vote amount.
  - **snapshot number**: round number.
  - **voters count**: how many voters.
  - **snapshot start timestamp**: start time.
  - **snapshot end timestamp**: end time.
  - **votes amount**: total votes(excluding withdraws).

GetVotingRecord

```
rpc GetVotingRecord (aelf.Hash) returns (VotingRecord){
}

message Hash{
```

(continues on next page)

(continued from previous page)

```
    bytes value = 1;
}

message VotingRecord {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp withdraw_timestamp = 5;
    google.protobuf.Timestamp vote_timestamp = 6;
    bool is_withdrawn = 7;
    string option = 8;
    bool is_change_target = 9;
}
```

Get the voting record for the given record ID.

- **Hash**
  - **value**: transaction id.
- **Returns**
  - **voting item id**: voting activity id.
  - **voter**: voter's address.
  - **snapshot number**: round number.
  - **withdraw timestamp**: withdraw time.
  - **vote timestamp**: vote time.
  - **is withdrawn**: indicate whether the vote has been withdrawn.
  - **option**: candidate id.
  - **is change target**: has withdrawn and vote to others.

GetVotingRecords

```
rpc GetVotingRecords (GetVotingRecordsInput) returns (VotingRecords){}

message GetVotingRecordsInput {
    repeated aelf.Hash ids = 1;
}

message Hash
{
    bytes value = 1;
}

message VotingRecords {
    repeated VotingRecord records = 1;
}

message VotingRecord {
    aelf.Hash voting_item_id = 1;
    aelf.Address voter = 2;
```

(continues on next page)

(continued from previous page)

```
    sint64 snapshot_number = 3;
    sint64 amount = 4;
    google.protobuf.Timestamp withdraw_timestamp = 5;
    google.protobuf.Timestamp vote_timestamp = 6;
    bool is_withdrawn = 7;
    string option = 8;
    bool is_change_target = 9;
}
```

Get the voting records for the given record IDs.

- **GetVotingRecordsInput**
  - **ids**: transaction ids.
- **Hash**
  - **value**: transaction id.
- **Returns**
  - **records**: records.
- **VotingRecord**
  - **voting item id**: voting activity id.
  - **voter**: voter's address.
  - **snapshot number**: round number.
  - **withdraw timestamp**: withdraw time.
  - **vote timestamp**: vote time.
  - **is withdrawn**: indicates whether the vote has been withdrawn.
  - **option**: candidate id.
  - **is change target**: has withdrawn and vote to others.

GetVotedItems

```
rpc GetVotedItems (aelf.Address) returns (VotedItems){
}

message Address{
    bytes value = 1;
}

message VotedItems {
    map<string, VotedIds> voted_item_vote_ids = 1;
}

message VotedIds {
    repeated aelf.Hash active_votes = 1;
    repeated aelf.Hash withdrawn_votes = 2;
}
```

Get the voter's withdrawn and valid transaction ids respectively.

- **Address**

  - **value**: voter's address.
- **Returns**
  - **voted item vote ids**: voting activity id => vote information.
- **VotedIds**
  - **active votes**: valid transaction id.
  - **withdrawn votes**: withdrawn transaction id.

GetVotingIds

```
rpc GetVotingIds (GetVotingIdsInput) returns (VotedIds){
}

message GetVotingIdsInput {
    aelf.Address voter = 1;
    aelf.Hash voting_item_id = 2;
}

message VotedIds {
    repeated aelf.Hash active_votes = 1;
    repeated aelf.Hash withdrawn_votes = 2;
}
```

Get the voter's withdrawn and valid transaction ids respectively according to voting activity id.

- **GetVotingIdsInput**
  - **voter**: voter's address.
  - **voting item id**: voting activity id.
- **Returns**
  - **active votes**: valid transaction id.
  - **withdrawn votes**: withdrawn transaction id.

19.12 Token Holder Contract

The TokenHolder contract is essentially used for building a bouns model for distributing bonus' to whom hold the token.

19.12.1 Actions

CreateScheme

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

message CreateTokenHolderProfitSchemeInput {
    string symbol = 1;
    sint64 minimum_lock_minutes = 2;
```

(continues on next page)

(continued from previous page)

```
    map<string, sint64> auto_distribute_threshold = 3;
}
```

It creates a scheme which stores relevant information about dividend in profit contract.

- **CreateTokenHolderProfitSchemeInput**
  - **symbol**: the token that will be used for locking and distributing profits.
  - **minimum** lock time: minimum lock time before withdrawing.
  - **automatic distribution threshold**: used when registering for profits (RegisterForProfits).

AddBeneficiary

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

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

Add a Beneficiary to a scheme.

- **AddTokenHolderBeneficiaryInput**
  - **beneficiary**: the new beneficiary.
  - **shares**: the shares to attribute to this beneficiary.

RemoveBeneficiary

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

message RemoveTokenHolderBeneficiaryInput {
    aelf.Address beneficiary = 1;
    sint64 amount = 2;
}
```

Remove a Beneficiary from a scheme. Note: this method can be used to remove a beneficiary or update its shares.

- **RemoveTokenHolderBeneficiaryInput**
  - **beneficiary**: the beneficiary to remove or update.
  - **amount**: 0 to remove the beneficiary. A positive integer, smaller than the current shares.

ContributeProfits

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

message ContributeProfitsInput {
    aelf.Address scheme_manager = 1;
    sint64 amount = 2;
```

(continues on next page)

(continued from previous page)

```
    string symbol = 3;
}
```

Contribute some token to a scheme.

- **ContributeProfitsInput**
  - **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
  - **amount**: the amount of tokens to contribute.
  - **symbol**: the token to contribute.

DistributeProfits

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

message DistributeProfitsInput {
    aelf.Address scheme_manager = 1;
    string symbol = 2;
}
```

Distribute a profit plan, which means its beneficiaries are going to get the shares.

- **DistributeProfitsInput**
  - **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
  - **symbol**: the token to contribute.

RegisterForProfits

```
rpc RegisterForProfits (RegisterForProfitsInput) returns (google.protobuf.Empty){}

message RegisterForProfitsInput {
    aelf.Address scheme_manager = 1;
    sint64 amount = 2;
}
```

- **RegisterForProfitsInput**
  - **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
  - **amount**: the amount of tokens to lock (and will correspond to the amount of shares).

Withdraw

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

This method will withdraw the given address for the Token Holder contract, this will also unlock the previously locked tokens.

ClaimProfits

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

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

The sender withdraws his/her token from the scheme.

- **ClaimProfitsInput**
  - **scheme manager**: manager of the scheme; when creating the scheme the Sender is set to manager.
  - **beneficiary**: the beneficiary, defaults to the Sender.
  - **symbol**: the symbol to claim.

19.12.2 View methods

GetScheme

```
rpc GetScheme (aelf.Address) returns (TokenHolderProfitScheme) { }

message TokenHolderProfitScheme {
    string symbol = 1;
    aelf.Hash scheme_id = 2;
    sint64 period = 3;
    sint64 minimum_lock_minutes = 4;
    map<string, sint64> auto_distribute_threshold = 5;
}
```

Returns a description of the scheme, wrapped in a **TokenHolderProfitScheme** object:

- **Returns**
  - **symbol**: the scheme's token.
  - **scheme id**: the id of the scheme.
  - **period**: the current period of the scheme.
  - **minimum lock minutes**: minimum lock time.
  - **automatic distribution threshold**: distribution info.

19.13 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.

19.13.1 Actions

InitialEconomicSystem

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

message InitialEconomicSystemInput {
    string native_token_symbol = 1;
    string native_token_name = 2;
    int64 native_token_total_supply = 3;
    int32 native_token_decimals = 4;
    bool is_native_token_burnable = 5;
    int64 mining_reward_total_amount = 6;
    int64 transaction_size_fee_unit_price = 7;
}
```

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.

- **IssueNativeTokenInput**
  - **native token symbol**: the native token symbol.
  - **native token name**: the native token name.
  - **native token total supply**: the native token total supply.
  - **native token decimals**: the token calculation is accurated to which decimal.
  - **is native token burnable**: it indicaites if the token is burnable.
  - **mining reward total amount**: It determines how much native token is used to reward the miners.
  - **transaction size fee unit price**: the transaction fee = transaction size * unit fee.

IssueNativeToken

```
rpc IssueNativeToken (IssueNativeTokenInput) returns (google.protobuf.Empty) {}

message IssueNativeTokenInput {
    int64 amount = 1;
    string memo = 2;
    aelf.Address to = 3;
}
```

Only ZeroContract is able to issue the native token.

- **IssueNativeTokenInput**
  - **amount**: amount of token.
  - **memo**: memo.
  - **to**: the recipient of the token.

19.13.2 Acs1 specific methods

For reference, you can find here the methods implementing acs1.

SetMethodFee

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

It sets method fee.

- **MethodFees**
  - **method name**: the method name in this contract.
  - **fees**: fee list.
- **MethodFee**
  - **symbol**: token symbol.
  - **basic fee**: fee.

ChangeMethodFeeController

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

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

Change the method fee controller, the default is Parliament.

- **AuthorityInfo**
  - **contract address**: new controller's contract address.
  - **owner address**: new controller's address.

GetMethodFee

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

This mehtod is used to query the method fee information.

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

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

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

This mehtod is used to query the controller of the method fee.

note: *for AuthorityInfo see ChangeMethodFeeController*

19.14 TokenConvert Contract

Using this contract can build a connection between the base token(the default is native token) and other tokens created on the chain. After building the connection, users can trade tokens with the Bancor model. You can find the detail information about Bancor in AElf Economic System White Paper.

19.14.1 Actions

InitializeInput

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

message InitializeInput {
    string base_token_symbol = 1;
    string fee_rate = 2;
    repeated Connector connectors = 3;
}

message Connector {
    string symbol = 1;
    int64 virtual_balance = 2;
    string weight = 3;
    bool is_virtual_balance_enabled = 4;
    bool is_purchase_enabled = 5;
    string related_symbol = 6;
    bool is_deposit_account = 7;
}
```

This method is used to initialize this contract (add base token, connections, etc.).

- **InitializeInput**
  - **base token symbol**: base token, default is the native token.
  - **fee rate**: buy/sell token need pay the fee(= cost * feeRate).
  - **connectors**: the default added connectors.

We use Bancor model to build the connection between base token and other tokens. Each pair connectors include the coefficients used by calculating the token price based on the base token, and it consists of the base token connector and the new token connector.

- **Connector**

- **symbol**: the connector symbol.
- **related symbol**: indicates its related connector, the pair connector includes a new created token connector and the base token connector.
- **virtual balance**: used in bancor model.
- **weight**: the weight is linked to the related connector's weight.
- **is virtual balance enabled**: true indicates that the virtual balance is used in price calculation.
- **is purchase enabled**: after build a pair connector, you can not buy/sell the token immediately, the default is false.
- **is deposit account**: indicates if the connector is base token connector.

AddPairConnector

```
rpc AddPairConnector(PairConnectorParam) returns (google.protobuf.Empty){}

message PairConnectorParam {
    string resource_connector_symbol = 1;
    string resource_weight = 2;
    int64 native_virtual_balance = 3;
    string native_weight = 4;
}
```

With Bancor model, each new token need a pair connectors to calculate its price. Only the connector contoller(the default is parliament) is allowed to call this API.

- **PairConnectorParam**
  - **resource connector symbol**: the new token connector's symbol.
  - **resource weight**: the new token connector's weight.
  - **native virtual balance**: the related base token connector's virtual balance.
  - **native weight**: base token's weight.

EnableConnector

```
rpc EnableConnector (ToBeConnectedTokenInfo) returns (google.protobuf.Empty){}

message ToBeConnectedTokenInfo{
    string token_symbol = 1;
    int64 amount_to_token_convert = 2;
}
```

To make the connection work, the connector contoller need send this transaction.

- **ToBeConnectedTokenInfo**
  - **token symbol**: the token symbol.
  - **amount to token convert**: to make the token trade, maybe you need deposit some base token.

UpdateConnector

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

Before calling the EnableConnector, the connector contoller update the pair connctors' information.

note: *for Connector see Initialize*

Buy

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

message BuyInput {
    string symbol = 1;
    int64 amount = 2;
    int64 pay_limit = 3;
}
```

After building the connection and enabling it, you can buy the new token with the base token.

- **BuyInput**
  - **symbol**: the token symbol.
  - **amount**: the amount you want to buy.
  - **pay limit**: no buy if paying more than this, 0 if no limit.

Sell

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

message SellInput {
    string symbol = 1;
    int64 amount = 2;
    int64 receive_limit = 3;
}
```

After building the connection and enabling it, you can sell the new token.

- **SellInput**
  - **symbol**: the token symbol.
  - **amount**: the amount you want to sell.
  - **receive limit**: no sell if receiving less than this, 0 if no limit.

ChangeConnectorController

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

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

The controller can be transferred to others.

note: *for AuthorityInfo see ChangeMethodFeeController*

19.14.2 Acs1 specific methods

For reference, you can find here the methods implementing acs1.

SetMethodFee

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

It sets method fee.

- **MethodFees**
  - **method name**: the method name in this contract.
  - **fees**: fee list.
- **MethodFee**
  - **symbol** token symbol.
  - **basic fee** fee.

ChangeMethodFeeController

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

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

Change the method fee controller, the default is Parliament.

- **AuthorityInfo**
  - **contract address**: new controller's contract address.
  - **owner address**: new controller's address.

GetMethodFee

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

This mehtod is used to query the method fee information.

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

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

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

This mehtod is used to query the controller of the method fee.

note: *for AuthorityInfo see ChangeMethodFeeController*

19.14.3 View methods

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

GetFeeReceiverAddress

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

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

This method is used to query the fee receiver.

- **Returns**
  - **value** the receiver's address, the half fee is burned and the other half is transferred to receiver.

GetControllerForManageConnector

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

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

his method is used to query the controller of the connector.

note: *for AuthorityInfo see ChangeConnectorController*

GetPairConnector

```
rpc GetPairConnector (TokenSymbol) returns (PairConnector){}

message TokenSymbol {
    string symbol = 1;
}

message PairConnector{
    Connector resource_connector = 1;
    Connector deposit_connector = 2;
}
```

This method is used to get the connection information between the base token and other token, which inlcudes a pair connctors.

- **TokenSymbol**
  - **symbol**: the token symbol.
- **Returns**
  - **resource connector**: the new add token connector.
  - **deposit connector**: the corresponding base token connector.

note: *for Connector see Initialize*

GetFeeRate

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

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

This method is used to query the fee rate.

- **Returns**
  - **value**: the fee rate.

GetBaseTokenSymbol

```
rpc GetBaseTokenSymbol (google.protobuf.Empty) returns (TokenSymbol){}

message TokenSymbol {
  string symbol = 1;
}
```

This method is used to query the base token symbol.

- **Returns**:

   - **symbol**: the token symbol.

GetBaseGetNeededDeposit

```
rpc GetNeededDeposit(ToBeConnectedTokenInfo) returns (DepositInfo){}

message ToBeConnectedTokenInfo{
    string token_symbol = 1;
    int64 amount_to_token_convert = 2;
}

message DepositInfo{
    int64 need_amount = 1;
    int64 amount_out_of_token_convert = 2;
}
```

This method is used to query how much the base token need be deposited before enabling the connectors.

- **ToBeConnectedTokenInfo**
   - **token symbol**: the token symbol.
   - **amount to token convert**: the added token amount you decide to transfer to TokenConvert.
- **Returns**
   - **need amount**: besides the amount you transfer to TokenConvert, how much base token you need deposit.
   - **amount out of token convert**: how much the added token have not transferred to TokenConvert.

GetDepositConnectorBalance

```
rpc GetDepositConnectorBalance(google.protobuf.StringValue) returns (google.protobuf.
↪Int64Value){}

message StringValue {
  string value = 1;
}

message Int64Value {
  int64 value = 1;
}
```

This method is used to query how much the base token have been deposited.

- **StringValue**:
   - **value**: the token symbol.
- **Returns**:
   - **value**: indicates for this token how much the base token have been deposited.

19.15 Configuration Contract

This contract's controller(the default is parliament) is able to save data(configuration) on the block chain.

19.15.1 Actions

SetConfiguration

```
rpc SetConfiguration (SetConfigurationInput) returns (google.protobuf.Empty){}

message SetConfigurationInput {
    string key = 1;
    bytes value = 2;
}
```

This method is used to add or reset configurations.

- **SetConfigurationInput**
  - **key**: the configuration's key.
  - **value**: the configuration's value(bianry data).

ChangeConfigurationController

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

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

The controller can be transfer to others.

- **AuthorityInfo**
  - **contract address**: new controller's contract address.
  - **owner address**: new controller's address.

19.15.2 Acs1 specific methods

For reference, you can find here the methods implementing acs1.

SetMethodFee

```
rpc SetMethodFee (MethodFees) returns (google.protobuf.Empty){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}

message MethodFee {
    string symbol = 1;
    int64 basic_fee = 2;
}
```

It sets method fee.

- **MethodFees**
  - **omethod name**: the method name in this contract.
  - **fees**: fee list.
- **MethodFee**:
  - **symbol**: token symbol.
  - **basic fee**: fee.

ChangeMethodFeeController

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

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

Change the method fee controller, the default is Parliament.

note: *for AuthorityInfo see ChangeConfigurationController*

GetMethodFee

```
rpc GetMethodFee (google.protobuf.StringValue) returns (MethodFees){}

message MethodFees {
    string method_name = 1;
    repeated MethodFee fees = 2;
}
```

This mehtod is used to query the method fee information.

note: *for MethodFees see SetMethodFee*

GetMethodFeeController

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

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

This mehtod is used to query the method fee information.

note: *for AuthorityInfo see ChangeMethodFeeController*

19.15.3 View methods

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

GetConfiguration

```
rpc GetConfiguration (google.protobuf.StringValue) returns (google.protobuf.
↪BytesValue){}

message StringValue {
  string value = 1;
}

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

This method is used to query a configurations.

- **StringValue**
  - **value**: the configuration's key.
- **Returns**
  - **value**: the configuration's data(bianry).

GetConfigurationController

This method is used to query the controller information.

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

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

- **Returns**
  - **contract address**: new controller's contract address.
  - **owner address**: new controller's address.

CHAPTER 20

Acs Introduction

20.1 ACS1 - Transaction Fee Standard

ACS1 is used to manage the transfer fee.

20.1.1 Interface

The contract inherited from ACS1 need implement the APIs below:

- SetMethodFee, the parameter type MethodFees defined in acs1.proto indicates the method name and its fee.
- GetMethodFee is used to get the method fee according to your input(method name).
- ChangeMethodFeeController is used to set who has the ablitity to call SetMethodFee, and its parameter's type AuthorityInfo is defined in authority_info.proto.
- GetMehodFeeController is used to get who has the ablitity to call SetMethodFee.

Attention: just the system contract on main chain is able to implement acs1.

20.1.2 Usage

On AElf main chain, before a transactoin start to execute, a pre-transaction is generated by pre-plugin FeeChargePre-ExecutionPlugin. 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>
```

(continues on next page)

(continued from previous page)

```
/// <returns></returns>
public override BoolValue ChargeTransactionFees(ChargeTransactionFeesInput input)
{
    // ...
    // Record tx fee bill during current charging process.
    var bill = new TransactionFeeBill();
    var fromAddress = Context.Sender;
    var methodFees = Context.Call<MethodFees>(input.ContractAddress,␣
↪nameof(GetMethodFee),
        new StringValue {Value = input.MethodName});
    var successToChargeBaseFee = true;
    if (methodFees != null && methodFees.Fees.Any())
    {
        successToChargeBaseFee = ChargeBaseFee(GetBaseFeeDictionary(methodFees), ref␣
↪bill);
    }
    var successToChargeSizeFee = true;
    if (!IsMethodFeeSetToZero(methodFees))
    {
        // Then also do not charge size fee.
        successToChargeSizeFee = ChargeSizeFee(input, ref bill);
    }
    // Update balances.
    foreach (var tokenToAmount in bill.FeesMap)
    {
        ModifyBalance(fromAddress, tokenToAmount.Key, -tokenToAmount.Value);
        Context.Fire(new TransactionFeeCharged
        {
            Symbol = tokenToAmount.Key,
            Amount = tokenToAmount.Value
        });
        if (tokenToAmount.Value == 0)
        {
            //Context.LogDebug(() => $"Maybe incorrect charged tx fee of
↪{tokenToAmount.Key}: it's 0.");
        }
    }
    return new BoolValue {Value = successToChargeBaseFee && successToChargeSizeFee};
}
```

In this method, the transaction fee consists of two parts:

1. The system calls GetMethodFee(line 15) to get the transacion fee you should pay. Then, it will check whether your balance is enough. If your balance is sufficient, the fee will be signed in the bill (variant bill). If not, your transaction will be rejected.
2. If the method fee is not set to 0 by the contract developer, the system will charge size fee. (the size if calculate by the paramter's size)

After charging successfully, an TransactionFeeCharged event is thrown, and the balance of the sender is modified.

The TransactionFeeCharged event will be captured and processed on the chain to calculate the total amount of transaction fees charged in the block. In the next block, the 10% of the transaction fee charged in this block is destroyed, the remaining 90% flows to dividend pool on the main chain, and is transferred to the FeeReciever on the side chain. The code is:

```
/// <summary>
```

(continues on next page)

(continued from previous page)

```
/// Burn 10% of tx fees.
/// If Side Chain didn't set FeeReceiver, burn all.
/// </summary>
/// <param name="symbol"></param>
/// <param name="totalAmount"></param>
private void TransferTransactionFeesToFeeReceiver(string symbol, long totalAmount)
{
    Context.LogDebug(() => "Transfer transaction fee to receiver.");
    if (totalAmount <= 0) return;
    var burnAmount = totalAmount.Div(10);
    if (burnAmount > 0)
        Context.SendInline(Context.Self, nameof(Burn), new BurnInput
        {
            Symbol = symbol,
            Amount = burnAmount
        });
    var transferAmount = totalAmount.Sub(burnAmount);
    if (transferAmount == 0)
        return;
    var treasuryContractAddress =
        Context.GetContractAddressByName(SmartContractConstants.
↪TreasuryContractSystemName);
    if ( treasuryContractAddress!= null)
    {
        // Main chain would donate tx fees to dividend pool.
        if (State.DividendPoolContract.Value == null)
            State.DividendPoolContract.Value = treasuryContractAddress;
        State.DividendPoolContract.Donate.Send(new DonateInput
        {
            Symbol = symbol,
            Amount = transferAmount
        });
    }
    else
    {
        if (State.FeeReceiver.Value != null)
        {
            Context.SendInline(Context.Self, nameof(Transfer), new TransferInput
            {
                To = State.FeeReceiver.Value,
                Symbol = symbol,
                Amount = transferAmount,
            });
        }
        else
        {
            // Burn all!
            Context.SendInline(Context.Self, nameof(Burn), new BurnInput
            {
                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.

20.1.3 Implementation

The easiest way to do this is to just implement the method GetMethodFee.

If there are Foo1, Foo2, Bar1 and Bar2 methods related to business logic in a contract, they are priced as 1, 1, 2, 2 ELF respectively, and the transaction fees of these four methods will not be easily modified later, they can be implemented as follows:

```
public override MethodFees GetMethodFee(StringValue input)
{
    if (input.Value == nameof(Foo1) || input.Value == nameof(Foo2))
    {
        return new MethodFees
        {
            MethodName = input.Value,
            Fees =
            {
                new MethodFee
                {
                    BasicFee = 1_00000000,
                    Symbol = Context.Variables.NativeSymbol
                }
            }
        };
    }
    if (input.Value == nameof(Bar1) || input.Value == nameof(Bar2))
    {
        return new MethodFees
        {
            MethodName = input.Value,
            Fees =
            {
                new MethodFee
                {
                    BasicFee = 2_00000000,
                    Symbol = Context.Variables.NativeSymbol
                }
            }
        };
    }
    return new MethodFees();
}
```

This implementation can modify the transaction fee only by upgrading the contract, without implementing the other three interfaces.

A more recommended implementation needs to define an MappedState in the State file for the contract:

```
public MappedState<string, MethodFees> TransactionFees { get; set; }
```

Modify the TransactionFees data structure in the SetMethodFee method, and return the value in the GetMethodFee method.

In this solution, the implementation of GetMethodFee is very easy:

```
public override MethodFees GetMethodFee(StringValue input)
    return State.TransactionFees[input.Value];
}
```

The implementation of SetMethodFee requires the addition of permission management, since contract developers don't want the transaction fees of their contract methods to be arbitrarily modified by others.

Referring to the MultiToken contract, it can be implemented as follows:

Firstly, define a SingletonState AuthorityInf(the AuthorityInf is defined in authority_info.proto)

```
public SingletonState<AuthorityInfo> MethodFeeController { get; set; }
```

Then, check the sender's right by comparing its address with OwnerAdress.

```
public override Empty SetMethodFee(MethodFees input)
{
  foreach (var symbolToAmount in input.Fees)
  {
     AssertValidToken(symbolToAmount.Symbol, symbolToAmount.BasicFee);
  }
  RequiredMethodFeeControllerSet();
  Assert(Context.Sender ==             State.MethodFeeController.Value.OwnerAddress,
↪"Unauthorized to set method fee.");
    State.TransactionFees[input.MethodName] = input;
    return new Empty();
}
```

AssertValidToken checks if the token symbol exists, and the BasicFee is reasonable.

The permission check code is in the lines 8 and 9, and RequiredMethodFeeControllerSet prevents the permission is not set before.

If permissions are not set, the SetMethodFee method can only be called by the default address of the Parliamentary contract. If a method is sent through the default address of Parliament, it means that two-thirds of the block producers have agreed to the proposal.

```
private void RequiredMethodFeeControllerSet()
{
   if (State.MethodFeeController.Value != null) return;
   if (State.ParliamentContract.Value == null)
   {
     State.ParliamentContract.Value =         Context.
↪GetContractAddressByName(SmartContractConstants.ParliamentContractSystemName);
   }
   var defaultAuthority = new AuthorityInfo();
   // Parliament Auth Contract maybe not deployed.
   if (State.ParliamentContract.Value != null)
   {
     defaultAuthority.OwnerAddress =               State.ParliamentContract.
↪GetDefaultOrganizationAddress.Call(new Empty());
     defaultAuthority.ContractAddress = State.ParliamentContract.Value;
   }
   State.MethodFeeController.Value = defaultAuthority;
}
```

Of course, the authority of SetMethodFee can also be changed, provided that the transaction to modify the authority is sent from the default address of the Parliamentary contract:

```
public override Empty ChangeMethodFeeController(AuthorityInfo input)
{
    RequiredMethodFeeControllerSet();
    AssertSenderAddressWith(State.MethodFeeController.Value.OwnerAddress);
    var organizationExist = CheckOrganizationExist(input);
    Assert(organizationExist, "Invalid authority input.");
    State.MethodFeeController.Value = input;
    return new Empty();
}
```

The implementation of GetMethodFeeController is also very easy

```
public override AuthorityInfo GetMethodFeeController(Empty input)
{
    RequiredMethodFeeControllerSet();
    return State.MethodFeeController.Value;
}
```

Above all, these are the two ways to implement acs1. Mostly, implementations will use a mixture of the two: part of methods' fee is set to a fixed value, the other part of method is not set method fee.

20.1.4 Test

Create ACS1's Stub, and call GetMethodFee and GetMethodFeeController to check if the return value is expected.

20.1.5 Example

All AElf system contracts implement ACS1, which can be used as a reference.

20.2 ACS2 - Parallel Execution Standard

ACS2 is used to provide information for parallel execution of transactions.

20.2.1 Interface

A contract that inherits ACS2 only needs to implement one method:

- GetResourceInfo

The parameter is the Transaction type, and the return value is the type ResourceInfo defined in acs2.proto:

```
message ResourceInfo {
    repeated aelf.ScopedStatePath paths = 1;
    bool non_parallelizable = 2;
}
```

aelf.ScopedStatePath is defined in aelf\core.proto:

```
message ScopedStatePath {
    Address address = 1;
    StatePath path = 2;
}
```

(continues on next page)

(continued from previous page)

```
message StatePath {
    repeated string parts = 1;
}
```

20.2.2 Usage

AElf uses the key-value database to store data. For the data generated during the contract execution, a mechanism called State Path is used to determine the key of the data.

For example Token contract's State file defines a property MappedState < Address, string, long >Balances, it can be used to access, modify balance.

Assuming that the address of the Token contract is **Nmjj7noTpMqZ522j76SDsFLhiKkThv1u3d4TxqJMD8v89tWmE**. If you want to know the balance of the address **2EM5uV6bSJh6xJfZTUa1pZpYsYcCUAdPvZvFUJzMDJEx3rbioz**, you can directly use this key to access redis / ssdb to get its value.

```
Nmjj7noTpMqZ522j76SDsFLhiKkThv1u3d4TxqJMD8v89tWmE/Balances/
↪2EM5uV6bSJh6xJfZTUa1pZpYsYcCUAdPvZvFUJzMDJEx3rbioz/ELF
```

On AElf, the implementation of parallel transaction execution is also based on the key , developers need to provide a method may access to the StatePath, then the corresponding transactions will be properly grouped before executing: if the two methods do not access the same StatePath, then you can safely place them in different groups.

Attention: The transaction will be canceled and labeled to "can not be groupped" when the StatePath mismatchs the method.

If you are interested in the logic, you can view the code ITransactionGrouper, as well as IParallelTransactionExecutingService .

20.2.3 Implementation

A example: within the Token contract, the core logic of method Transfer is to modify the balance of address. It accesses the balances property mentioned above twice.

At this point, we need to notify ITransactionGrouper via the GetResourceInfo method of the key of the ELF balance of address A and address B:

```
var args = TransferInput.Parser.ParseFrom(txn.Params);
var resourceInfo = new ResourceInfo
{
    Paths =
    {
        GetPath(nameof(TokenContractState.Balances), txn.From.ToString(), args.
↪Symbol),
        GetPath(nameof(TokenContractState.Balances), args.To.ToString(), args.Symbol),
    }
};
return resourceInfo;
```

The GetPath forms a ScopedStatePath from several pieces of data that make up the key:

```
private ScopedStatePath GetPath(params string[] parts)
{
    return new ScopedStatePath
```

(continues on next page)

(continued from previous page)

```
    {
        Address = Context.Self,
        Path = new StatePath
        {
            Parts =
            {
                parts
            }
        }
    }
}
```

20.2.4 Test

You can construct two transactions, and the transactions are passed directly to an implementation instance of ITransactionGrouper, and the GroupAsync method is used to see if the two transactions are parallel.

We prepare two stubs that implement the ACS2 contract with different addresses to simulate the Transfer:

```
var keyPair1 = SampleECKeyPairs.KeyPairs[0];
var acs2DemoContractStub1 = GetACS2DemoContractStub(keyPair1);
var keyPair2 = SampleECKeyPairs.KeyPairs[1];
var acs2DemoContractStub2 = GetACS2DemoContractStub(keyPair2);
```

Then take out some services and data needed for testing from Application:

```
var transactionGrouper = Application.ServiceProvider.GetRequiredService
↪<ITransactionGrouper>();
var blockchainService = Application.ServiceProvider.GetRequiredService
↪<IBlockchainService>();
var chain = await blockchainService.GetChainAsync();
```

Finally, check it via transactionGrouper:

```
// Situation can be parallel executed.
{
    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
        }),
    });
```

(continues on next page)

(continued from previous page)

```
    groupedTransactions.Parallelizables.Count.ShouldBe(2);
}
// Situation cannot.
{
    var groupedTransactions = await transactionGrouper.GroupAsync(new ChainContext
    {
        BlockHash = chain.BestChainHash,
        BlockHeight = chain.BestChainHeight
    }, new List<Transaction>
    {
        acs2DemoContractStub1.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
        acs2DemoContractStub2.TransferCredits.GetTransaction(new TransferCreditsInput
        {
            To = Address.FromPublicKey(SampleECKeyPairs.KeyPairs[2].PublicKey),
            Symbol = "ELF",
            Amount = 1
        }),
    });
    groupedTransactions.Parallelizables.Count.ShouldBe(1);
}
```

20.2.5 Example

You can refer to the implementation of the MultiToken contract for GetResourceInfo. Noting that for the ResourceInfo provided by the method Tranfer, you need to consider charging a transaction fee in addition to the two keys mentioned in this article.

20.3 ACS3 - Contract Proposal Standard

Using the AuthorityInfo defined in authority_info.proto restricts a method to be called by a certain address:

```
Assert(Context.Sender == State.AuthorityInfo.Value.OwnerAddress, "No permission.");
```

When a method needs to be agreed by multiple parties, the above solution is obviously inadequate. At this time, you can consider using some of the interfaces provided by ACS3.

20.3.1 Interface

If you want multiple addresses vote to get agreement to do something, you can implement the following methods defined in ACS3:

- CreateProposal, it is to specify a method for a contract and its parameters. When the proposal is approved by multiple addresses, it can be released: use a virtual address as a Sender, and execute this method by sending an inline transaction. Therefore, the parameter CreateProposalInput defines the basic information of the inline transaction to be executed finally. The return value is a hash, which is used to uniquely identify this proposal;

- Approve, Reject, Abstain, the parameters are Hash, called the proposal Id, created by CreateProposal, is used to agree, reject, and abstain respectively .
- Release, the parameter is the proposal Id, is used to release the proposal: when the requirements are met, it can be released;
- ClearProposal is used to clean invalid data in DB.

It can be seen that before a proposal is released, the account with voting rights can agree, object, and abstain. Which specific accounts have the right to vote? ACS3 introduces the concept of Organization. A proposal is attached to an organization from its creation, and only members of the organization can vote.

However, due to the different forms of organization, the Organization structure needs to be defined by the contract implementing the ACS3. Here is an example:

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

Because each organization has a default virtual address, adding the code like the begining at this document can verify if the sender is authorized.

```
Assert(Context.Sender == someOrganization.OrganizationAddress, "No permission.");
```

How to know what an orgnanization has agreed on a proposal? ACS3 defines a data structure ProposalReleaseThreshold:

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

The orgnaization determines how to deal with the proposal according to the data:

- the minimal approval the proposal can be released.
- The most rejection amount the proposal can tolerate.
- The most abstention amount the proposal can tolerate.
- the minimal vote amount the proposal is valid.

Interfaces referencing organization in ACS3:

- ChangeOrganizationThresholdits paramenter is ProposalReleaseThreshold that is used to modify the threshold. Of course, this method also needs permission control
- ChangeOrganizationProposerWhiteList, The organization can restrict which addresses can create proposals. Its parameter is ProposerWhiteList, defined in acs3.proto, which is actually an Address list;
- CreateProposalBySystemContract, The original intention is that the system contract can create a proposal via the virtual address, that is, there are some senders have privileges, and must be a contract:

The type of APIs mentioned above is action, there are some APIs with type View used to query:

- GetProposal is used to get the proposal detailed information.

- ValidateOrganizationExist is used to check if the organization exists in a contract.
- ValidateProposerInWhiteList is used to check if the address is in the whitelist of a organization.

20.3.2 Implementation

It is assumed here that there is only one organization in a contract, that is, there is no need to specifically define the Organization type. Since the organization is not explicitly declared and created, the organization's proposal whitelist does not exist. The process here is that the voter must use a certain token to vote.

For simplicity, only the core methods CreateProposal, Approve, Reject, Abstain, and Release are implemented here.

There are only two necessary State attributes:

```
public MappedState<Hash, ProposalInfo> Proposals { get; set; }
public SingletonState<ProposalReleaseThreshold> ProposalReleaseThreshold { get; set; }
```

The Proposals stores all proposal's information, and the ProposalReleaseThreshold is used to save the requirements that the contract needs to meet to release the proposal.

When the contract is initialized, the proposal release requirements should be set:

```
public override Empty Initialize(Empty input)
{
    State.TokenContract.Value =
        Context.GetContractAddressByName(SmartContractConstants.
↪TokenContractSystemName);
    State.ProposalReleaseThreshold.Value = new ProposalReleaseThreshold
    {
        MinimalApprovalThreshold = 1,
        MinimalVoteThreshold = 1
    };
    return new Empty();
}
```

The requirement is at least one member who vote and at least one approval. Create proposal:

```
public override Hash CreateProposal(CreateProposalInput input)
{
    var proposalId = Context.GenerateId(Context.Self, input.Token);
    Assert(State.Proposals[proposalId] == null, "Proposal with same token already␣
↪exists.");
    State.Proposals[proposalId] = new ProposalInfo
    {
        ProposalId = proposalId,
        Proposer = Context.Sender,
        ContractMethodName = input.ContractMethodName,
        Params = input.Params,
        ExpiredTime = input.ExpiredTime,
        ToAddress = input.ToAddress,
        ProposalDescriptionUrl = input.ProposalDescriptionUrl
    };
    return proposalId;
}
```

Vote:

```
public override Empty Abstain(Hash input)
{
    Charge();
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    proposal.Abstentions.Add(Context.Sender);
    State.Proposals[input] = proposal;
    return new Empty();
}
public override Empty Approve(Hash input)
{
    Charge();
    var proposal = State.Proposals[input];
    if (proposal == null)
    {
        throw new AssertionException("Proposal not found.");
    }
    proposal.Approvals.Add(Context.Sender);
    State.Proposals[input] = proposal;
    return new Empty();
}
public override Empty Reject(Hash input)
{
    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.");
```

(continues on next page)

(continued from previous page)

```
    Context.SendInline(proposal.ToAddress, proposal.ContractMethodName, proposal.
↪Params);
    return new Empty();
}
private bool IsReleaseThresholdReached(ProposalInfo proposal)
{
    var isRejected = IsProposalRejected(proposal);
    if (isRejected)
        return false;
    var isAbstained = IsProposalAbstained(proposal);
    return !isAbstained && CheckEnoughVoteAndApprovals(proposal);
}
private bool IsProposalRejected(ProposalInfo proposal)
{
    var rejectionMemberCount = proposal.Rejections.Count;
    return rejectionMemberCount > State.ProposalReleaseThreshold.Value.
↪MaximalRejectionThreshold;
}
private bool IsProposalAbstained(ProposalInfo proposal)
{
    var abstentionMemberCount = proposal.Abstentions.Count;
    return abstentionMemberCount > State.ProposalReleaseThreshold.Value.
↪MaximalAbstentionThreshold;
}
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;
}
```

20.3.3 Test

Before testing, two methods were added to the contract, that had just implemented ACS3. We will test the proposal with these mehods.

Define a singleton string in the State file:

```
public StringState Slogan { get; set; }
```

Then implement a pair of Set/Get methods:

```
public override StringValue GetSlogan(Empty input)
{
    return State.Slogan.Value == null ? new StringValue() : new StringValue {Value =␣
↪State.Slogan.Value};
}
```

(continues on next page)

(continued from previous page)

```
public override Empty SetSlogan(StringValue input)
{
    Assert(Context.Sender == Context.Self, "No permission.");
    State.Slogan.Value = input.Value;
    return new Empty();
}
```

In this way, during the test, create a proposal for the SetSlogan. After passing and releasing, use the GetSlogan method to check whether the Slogan has been modified.

Prepare a Stub that implements the ACS3 contract:

```
var keyPair = SampleECKeyPairs.KeyPairs[0];
var acs3DemoContractStub =
    GetTester<ACS3DemoContractContainer.ACS3DemoContractStub>(DAppContractAddress,␣
↪keyPair);
```

Since approval requires the contract to charge users, the user should send Approve transaction of the Token contract.

```
var tokenContractStub =
    GetTester<TokenContractContainer.TokenContractStub>(
        GetAddress(TokenSmartContractAddressNameProvider.StringName), keyPair);
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = DAppContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
```

Create a proposal, the target method is SetSlogan, here we want to change the Slogan to “AElf” :

```
var proposalId = (await acs3DemoContractStub.CreateProposal.SendAsync(new␣
↪CreateProposalInput
{
    ContractMethodName = nameof(acs3DemoContractStub.SetSlogan),
    ToAddress = DAppContractAddress,
    ExpiredTime = TimestampHelper.GetUtcNow().AddHours(1),
    Params = new StringValue {Value = "AElf"}.ToByteString(),
    Token = HashHelper.ComputeFrom("AElf")
})).Output;
```

Make sure that the Slogan is still an empty string at this time and then vote:

```
// Check slogan
{
    var slogan = await acs3DemoContractStub.GetSlogan.CallAsync(new Empty());
    slogan.Value.ShouldBeEmpty();
}
await acs3DemoContractStub.Approve.SendAsync(proposalId);
```

Release proposal, and the Slogan becomes “AElf”.

```
await acs3DemoContractStub.Release.SendAsync(proposalId);
// Check slogan
{
    var slogan = await acs3DemoContractStub.GetSlogan.CallAsync(new Empty());
```

(continues on next page)

(continued from previous page)

```
    slogan.Value.ShouldBe("AElf");
}
```

20.4 ACS4 - Consensus Standard

ACS4 is used to customize consensus mechanisms.

20.4.1 Interface

If you want to customize the consensus mechanism, you need to implement the following five interfaces:

- GetConsensusCommand, whose parameter is a binary array, returns ConsensusCommand defined in acs4.proto. This type is used to indicate the start time of the next block, the block time limit, and the final cut-off time for the account calling GetConsensus Command;
- GetConsensusExtraData, the parameters and return values are binary arrays, which are used to generate consensus block header information through consensus contracts when a new block is produced;
- GenerateConsensusTransactions, the parameter is a binary array, and the return value is of type TransactionList. It is used to generate a consensus system transaction when a block is generated. Each block will contain only one consensus transaction, which is used to write the latest consensus information to the State database;
- ValidateConsensusBeforeExecution, the parameter is a binary array, and the return value is of type ValidationResult, is used to verify whether the consensus information in the block header is correct before the block executes;
- ValidateConsensusAfterExecution, with the same parameter and return value, is used to verify that the consensus information written to the State is correct after the block executes.

ConsensusCommand, ValidationResult and TransactionList are defined as:

```
message ConsensusCommand {
    int32 limit_milliseconds_of_mining_block = 1;// Time limit of mining next block.
    bytes hint = 2;// Context of Hint is diverse according to the consensus protocol␣
↪we choose, so we use bytes.
    google.protobuf.Timestamp arranged_mining_time = 3;
    google.protobuf.Timestamp mining_due_time = 4;
}
message ValidationResult {
    bool success = 1;
    string message = 2;
    bool is_re_trigger = 3;
}
message TransactionList {
    repeated aelf.Transaction transactions = 1;
}
```

20.4.2 Usage

The five interfaces defined in ACS4 basically correspond to the five methods of the IConsensusService interface in the AElf.Kernel.Consensus project:

ACS4	IConsensusService	Methodology	The Timing To Call
`GetConsensusCommand`	Task TriggerConsensusAsync (ChainContext chainContext);	When TriggerConsensusAsync is called, it will use the account configured by the node to call the GetConsensusCommand method of the consensus contract to obtain block information ConsensusCommand), and use it to (see IConsensusScheduler implementation) .	1. When the node is started; 2. When the BestChainFound-EventData 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);
`GetConsensus-ExtraData`	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.
`GenerateConsensus-Transactions`	Task<List<Transaction>> GenerateConsensus-TransactionsAsync(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.

20.4.3 Example

You can refer to the implementation of the AEDPoS contract.

20.5 ACS5 - Contract Threshold Standard

If you want to raise the threshold for using contract, consider implementing ACS5.

20.5.1 Interface

To limit to call a method in a contract, you only need to implement an interface:

- GetMethodCallingThreshold, the parameter is string, and the return value is the MethodCallingThreshold defined in the acs5.proto file.

If you want to modify the threshold after the contract is deployed, another interface can be implemented:

- SetMethodCallingThreshold, the parameter is SetMethodCallingThresholdInput.

The definition of MethodCallingThreshold type is:

```
message MethodCallingThreshold {
    map<string, int64> symbol_to_amount = 1;// The order matters.
    ThresholdCheckType threshold_check_type = 2;
}
enum ThresholdCheckType {
    BALANCE = 0;
    ALLOWANCE = 1;
}
```

The significance of the enumeration ThresholdCheckType is that there are two types of thresholds for contract method calls:

1. It can be called when the balance of a certain token in the account is sufficient, which corresponds to ThresholdCheckType.Balance;
2. Not only does the balance of a token in the account be required to be sufficient, but the account also needs sufficient authorization for the target contract, which corresponds to The ThresholdCheckType.Allowance.
3. SetMethodCallingThresholdInput definition

```
message SetMethodCallingThresholdInput {
    string method = 1;
    map<string, int64> symbol_to_amount = 2;// The order matters.
    ThresholdCheckType threshold_check_type = 3;
}
```

20.5.2 Usage

Similar to ACS1, which uses an automatically generated pre-plugin transaction called ChargeTransactionFees to charge a transaction fee, ACS5 automatically generates a pre-plugin transaction called CheckThreshold to test whether the account that sent the transaction can invoke the corresponding method.

The implementation of CheckThreshold:

```
public override Empty CheckThreshold(CheckThresholdInput input)
{
    var meetThreshold = false;
    var meetBalanceSymbolList = new List<string>();
    foreach (var symbolToThreshold in input.SymbolToThreshold)
    {
        if (GetBalance(input.Sender, symbolToThreshold.Key) < symbolToThreshold.Value)
            continue;
        meetBalanceSymbolList.Add(symbolToThreshold.Key);
    }
    if (meetBalanceSymbolList.Count > 0)
    {
        if (input.IsCheckAllowance)
        {
            foreach (var symbol in meetBalanceSymbolList)
            {
                if (State.Allowances[input.Sender][Context.Sender][symbol] <
                    input.SymbolToThreshold[symbol]) continue;
                meetThreshold = true;
                break;
            }
        }
        else
        {
            meetThreshold = true;
        }
    }
    if (input.SymbolToThreshold.Count == 0)
    {
        meetThreshold = true;
    }
    Assert(meetThreshold, "Cannot meet the calling threshold.");
    return new Empty();
}
```

In other words, if the token balance of the sender of the transaction or the amount authorized for the target contract does not reach the set limit, the pre-plugin transaction will throw an exception, thereby it prevents the original transaction from executing.

20.5.3 Implementation

As the GetMethodFee of ACS1, you can implement only one GetMethodCallingThreshold method.

It can also be achieved by using MappedState<string, MethodCallingThreshold> in the State file:

```
public MappedState<string, MethodCallingThreshold> MethodCallingThresholds { get; set;
↪ }
```

But at the same time, do not forget to configure the call permission of SetMethodCallingThreshold, which requires the definition of an Admin in the State (of course, you can also use ACS3):

```
public SingletonState<Address> Admin { get; set; }
```

The easiest implementation

```
public override Empty SetMethodCallingThreshold(SetMethodCallingThresholdInput input)
{
    Assert(State.Admin.Value == Context.Sender, "No permission.");
    State.MethodCallingThresholds[input.Method] = new MethodCallingThreshold
    {
        SymbolToAmount = {input.SymbolToAmount}
    };
    return new Empty();
}
public override MethodCallingThreshold GetMethodCallingThreshold(StringValue input)
{
    return State.MethodCallingThresholds[input.Value];
}
public override Empty Foo(Empty input)
{
    return new Empty();
}
```

20.5.4 Test

You can test the Foo method defined above.

Make a Stub:

```
var keyPair = SampleECKeyPairs.KeyPairs[0];
var acs5DemoContractStub =
    GetTester<ACS5DemoContractContainer.ACS5DemoContractStub>(DAppContractAddress,␣
↪keyPair);
```

Before setting the threshold, check the current threshold, which should be 0:

```
var methodResult = await acs5DemoContractStub.GetMethodCallingThreshold.CallAsync(
    new StringValue
    {
        Value = nameof(acs5DemoContractStub.Foo)
    });
methodResult.SymbolToAmount.Count.ShouldBe(0);
```

The ELF balance of the caller of Foo should be greater than 1 ELF:

```
await acs5DemoContractStub.SetMethodCallingThreshold.SendAsync(
    new SetMethodCallingThresholdInput
    {
        Method = nameof(acs5DemoContractStub.Foo),
        SymbolToAmount =
        {
            {"ELF", 1_0000_0000}
        },
        ThresholdCheckType = ThresholdCheckType.Balance
    });
```

Check the threshold again:

```
methodResult = await acs5DemoContractStub.GetMethodCallingThreshold.CallAsync(
    new StringValue
    {
```

(continues on next page)

(continued from previous page)

```
        Value = nameof(acs5DemoContractStub.Foo)
    });
methodResult.SymbolToAmount.Count.ShouldBe(1);
methodResult.ThresholdCheckType.ShouldBe(ThresholdCheckType.Balance);
```

Send the Foo transaction via an account who has sufficient balance can succeed:

```
// Call with enough balance.
{
    var executionResult = await acs5DemoContractStub.Foo.SendAsync(new Empty());
    executionResult.TransactionResult.Status.ShouldBe(TransactionResultStatus.Mined);
}
```

Send the Foo transaction via another account without ELF fails:

```
// Call without enough balance.
{
    var poorStub =
        GetTester<ACS5DemoContractContainer.ACS5DemoContractStub>(DAppContractAddress,
            SampleECKeyPairs.KeyPairs[1]);
    var executionResult = await poorStub.Foo.SendWithExceptionAsync(new Empty());
    executionResult.TransactionResult.Error.ShouldContain("Cannot meet the calling␣
↪threshold.");
}
```

20.6 todo

20.7 ACS7 - Contract CrossChain Standard

ACS7 is for cross chain related contract implementation.

20.7.1 Interface

This involves methods for chain creation and indexing:

- **ProposeCrossChainIndexing**
  - Propose cross chain to be indexed and wait for approvals from authorized organizations;
- **ReleaseCrossChainIndexing**
  - Release the proposed indexing if already approved.
- **RecordCrossChainData**
  - The method to record cross chain data and complete indexing.
- **RequestSideChainCreation**
  - Request to create a new side chain e wait for approvals from authorized organizations;.
- **ReleaseSideChainCreation**
  - Release the side chain creation request if already approved and it will call the method CreateSideChain.

- **CreateSideChain**
  - The method to create a new side chain.
- **Recharge**
  - Recharge for one specific side chain.
- **DisposeSideChain**
  - Stop indexing for one specific side chain.
- **AdjustIndexingFeePrice**
  - Adjust indexing fee for one specific side chain.
- **VerifyTransaction**
  - Transaction cross chain verification.
- **GetChainInitializationData**
  - Get side chain initialization data which is needed during the first time startup of side chain node.

20.7.2 Usage

ACS7 declares methods for the scenes about cross chain. AElf provides the implementation for ACS7, CrossChainContract. Please refer *api docs* of this contract for more details.

20.8 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.

20.8.1 Interface

Only one method is defined in the acs8.proto file:

- BuyResourceToken, the parameter is the BuyResourceTokenInput defined in the Proto file.

```
message BuyResourceTokenInput {
    string symbol = 1;
    int64 amount = 2;
    int64 pay_limit = 3; // No buy if paying more than this, 0 if no limit
}
```

This method can be used to purchase one of the four resource coins, which consumes the ELF balance in the contract account (you can recharge it yourself, or you can collect the user's ELF tokens as a profit to be self-sufficient).

Of course, it is possible for developers to purchase resource token and then directly transfer to the address of this contract via the Transfer method of the Token contract. Therefore, this interface does not have to be implemented.

20.8.2 Usage

The contract inherited from ACS1 uses a pre-plugin transaction called ChargeTransactionFees for charging transaction fee.

Because the specific charge amount is determined by the actual consumption of the transaction, the post-plugin generates ChargeResourceToken tansaction to charge resource token.

The implementation of ChargeResourceToken is also similar to it of ChargeTransactionFees:

```
public override Empty ChargeResourceToken(ChargeResourceTokenInput input)
{
    Context.LogDebug(() => string.Format("Start executing ChargeResourceToken.{0}",␣
↪input));
    if (input.Equals(new ChargeResourceTokenInput()))
    {
        return new Empty();
    }
    var bill = new TransactionFeeBill();
    foreach (var pair in input.CostDic)
    {
        Context.LogDebug(() => string.Format("Charging {0} {1} tokens.", pair.Value,␣
↪pair.Key));
        var existingBalance = GetBalance(Context.Sender, pair.Key);
        Assert(existingBalance >= pair.Value,
            string.Format("Insufficient resource of {0}. Need balance: {1}; Current␣
↪balance: {2}.", pair.Key, pair.Value, existingBalance));
        bill.FeesMap.Add(pair.Key, pair.Value);
    }
    foreach (var pair in bill.FeesMap)
    {
        Context.Fire(new ResourceTokenCharged
        {
            Symbol = pair.Key,
            Amount = pair.Value,
            ContractAddress = Context.Sender
        });
        if (pair.Value == 0)
        {
            Context.LogDebug(() => string.Format("Maybe incorrect charged resource␣
↪fee of {0}: it's 0.", pair.Key));
        }
    }
    return new Empty();
}
```

The amount of each resource token should be calculated by AElf.Kernel.FeeCalculation. In detail, A data structure named CalculateFeeCoefficients is defined in token_contract.proto, whose function is to save all coefficients of a polynomial, and every three coefficients are a group, such as a, b, c, which means (b / c) * x ^ a. Each resource token has a polynomial that calculates it. Then according to the polynomial and the actual consumption of the resource, calculate the cost of the resource token. Finally, the cost is used as the parameter of ChargeResourceToken to generate this post-plugin transaction.

In addition, the method of the contract that has been owed cannot be executed before the contract top up resource token. As a result, a pre-plugin transaction is added, similar to the ACS5 pre-plugin transaction, which checks the contract's resource token balance, and the transaction's method name is CheckResourceToken :

```
public override Empty CheckResourceToken(Empty input)
{
    foreach (var symbol in Context.Variables.GetStringArray(TokenContractConstants.
↪PayTxFeeSymbolListName))
    {
        var balance = GetBalance(Context.Sender, symbol);
        var owningBalance = State.OwningResourceToken[Context.Sender][symbol];
        Assert(balance > owningBalance,
            string.Format("Contract balance of {0} token is not enough. Owning {1}.",␣
↪symbol, owningBalance));
    }
    return new Empty();
}
```

20.9 ACS9 - Contract profit dividend standard

On the AElf's side chain, the contract needs to declare where its profits are going, and implemente ACS9.

20.9.1 Interface

ACS9 contains an method which does not have to be implemented:

- TakeContractProfits is used for the developer to collect the profits from the contract. and the profits will be distributed in this method. There are also other methods for developers to claim profits. However, these methods needs to be approved before deployment/upgrade.

Two View methods that must be implemented. They are mostly used in the AElf blockchain browser:

- GetProfitConfig, whose return value is the ProfitConfig defined in acs9.proto, includes the profit token symbol list, the token symbol that the user can lock them to claim the profit, and the portion of the profit that will be donated to the dividend pool each time the developer receives the profit. When reviewing the contract code, you should check if the same ProfitConfig data is actually used for distributing the profits.
- GetProfitsAmount, as the name implies, is used to query the profits of the contract so far, with a return value of type ProfitsMap.

ProfitConfig is defined as:

```
message ProfitConfig {
    int32 donation_parts_per_hundred = 1;
    repeated string profits_token_symbol_list = 2;
    string staking_token_symbol = 3;
}
```

The ProfitsMap type is essentially a map from token symbol to the amount:

```
message ProfitsMap {
    map<string, int64> value = 1;
}
```

20.9.2 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,
        Amount = ACS9DemoContractConstants.TotalSupply / 5,
        Symbol = State.Symbol.Value,
        Memo = "Issue token for profit receiver"
    });
}
```

(continues on next page)

(continued from previous page)

```
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.
```

(continues on next page)

(continued from previous page)

```
    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;
    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,
        Spender = State.DividendPoolContract.Value
    });
    State.DividendPoolContract.Donate.Send(new DonateInput
    {
```

(continues on next page)

(continued from previous page)

```
        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;
}
```

20.9.3 Test

Since part of the profits from the ACS9 contract transfer to the Token contract and the other transfer to the dividend pool, a TokenHolder Stub and a contract implementing ACS10 Stub are required in the test. Accordingly, the contracts that implements ACS9 or ACS10 need to be deployed. Before the test begins, the contract implementing ACS9 can be initialized by interface IContractInitializationProvider, and sets the dividend pool's name to the other contract's name:

```
public class ACS9DemoContractInitializationProvider : IContractInitializationProvider
{
    public List<InitializeMethod> GetInitializeMethodList(byte[] contractCode)
    {
        return new List<InitializeMethod>
        {
            new InitializeMethod
            {
                MethodName = nameof(ACS9DemoContract.Initialize),
                Params = new InitializeInput
                {
```

(continues on next page)

(continued from previous page)

```
                    ProfitReceiver = Address.FromPublicKey(SampleECKeyPairs.KeyPairs.
↪Skip(3).First().PublicKey),
                    DividendPoolContractName = ACS10DemoSmartContractNameProvider.Name
                }.ToByteString()
            }
        };
    }
    public Hash SystemSmartContractName { get; } = ACS9DemoSmartContractNameProvider.
↪Name;
    public string ContractCodeName { get; } = "AElf.Contracts.ACS9DemoContract";
}
```

Prepare a user account:

```
protected List<ECKeyPair> UserKeyPairs => SampleECKeyPairs.KeyPairs.Skip(2).Take(3).
↪ToList();
```

Prepare some Stubs:

```
var keyPair = UserKeyPairs[0];
var address = Address.FromPublicKey(keyPair.PublicKey);
// Prepare stubs.
var acs9DemoContractStub = GetACS9DemoContractStub(keyPair);
var acs10DemoContractStub = GetACS10DemoContractStub(keyPair);
var userTokenStub =
    GetTester<TokenContractImplContainer.TokenContractImplStub>(TokenContractAddress,␣
↪UserKeyPairs[0]);
var userTokenHolderStub =
    GetTester<TokenHolderContractContainer.TokenHolderContractStub>
↪(TokenHolderContractAddress,
        UserKeyPairs[0]);
```

Then, transfer ELF to the user (TokenContractStub is the Stub of the initial bp who has much ELF) :

```
// Transfer some ELFs to user.
await TokenContractStub.Transfer.SendAsync(new TransferInput
{
    To = address,
    Symbol = "ELF",
    Amount = 1000_00000000
});
```

Have the user call SignUp to check if he/she has got 10 APP tokens:

```
await acs9DemoContractStub.SignUp.SendAsync(new Empty());
// User has 10 APP tokens because of signing up.
(await GetFirstUserBalance("APP")).ShouldBe(10_00000000);
```

Test the recharge method of the contract itself:

```
var elfBalanceBefore = await GetFirstUserBalance("ELF");
// User has to Approve an amount of ELF tokens before deposit to the DApp.
await userTokenStub.Approve.SendAsync(new ApproveInput
{
    Amount = 1000_00000000,
    Spender = ACS9DemoContractAddress,
```

(continues on next page)

(continued from previous page)

```
    Symbol = "ELF"
});
await acs9DemoContractStub.Deposit.SendAsync(new DepositInput
{
    Amount = 100_00000000
});
// Check the change of balance of ELF.
var elfBalanceAfter = await GetFirstUserBalance("ELF");
elfBalanceAfter.ShouldBe(elfBalanceBefore - 100_00000000);
// Now user has 110 APP tokens.
(await GetFirstUserBalance("APP")).ShouldBe(110_00000000);
```

The user locks up 57 APP via the TokenHolder contract in order to obtain profits from the contract:

```
// User lock some APP tokens for getting profits. (APP -57)
await userTokenHolderStub.RegisterForProfits.SendAsync(new RegisterForProfitsInput
{
    SchemeManager = ACS9DemoContractAddress,
    Amount = 57_00000000
});
```

The Use method is invoked 10 times and 0.3 APP is consumed each time, and finally the user have 50 APP left:

```
await userTokenStub.Approve.SendAsync(new ApproveInput
{
    Amount = long.MaxValue,
    Spender = ACS9DemoContractAddress,
    Symbol = "APP"
});
// User uses 10 times of this DApp. (APP -3)
for (var i = 0; i < 10; i++)
{
    await acs9DemoContractStub.Use.SendAsync(new Record());
}
// Now user has 50 APP tokens.
(await GetFirstUserBalance("APP")).ShouldBe(50_00000000);
```

Using the TakeContractProfits method, the developer attempts to withdraw 10 ELF as profits. The 10 ELF will be transferred to the developer in this method:

```
const long baseBalance = 0;
{
    var balance = await TokenContractStub.GetBalance.CallAsync(new GetBalanceInput
    {
        Owner = UserAddresses[1], Symbol = "ELF"
    });
    balance.Balance.ShouldBe(baseBalance);
}
// Profits receiver claim 10 ELF profits.
await acs9DemoContractStub.TakeContractProfits.SendAsync(new TakeContractProfitsInput
{
    Symbol = "ELF",
    Amount = 10_0000_0000
});
// Then profits receiver should have 9.9 ELF tokens.
{
```

(continues on next page)

(continued from previous page)

```
    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",
    Owner = UserAddresses[0]
});
resultBalance.Balance.ShouldBe(beforeBalance.Balance + 57_00000000);
```

20.10 ACS10 - Dividend Pool Standard

ACS10 is used to construct a dividend pool in the contract.

20.10.1 Interface

To construct a dividend pool, you can implement the following interfaces optionally:

- Donate is used to donate dividend pool, parameters include the token symbol and the amount to be donated to the dividend pool;
- Release is used to release the dividend pool. The parameter is the number of sessions to release dividends. Be careful to set its calling permission.
- SetSymbolList is used to set the token symbols dividend pool supports. The parameter is of type SymbolList.
- GetSymbolList is used to get the token symbols dividend pool supports. The return type is SymbolList.
- GetUndistributdividends is used to obtain tokens' balance that have not been distributed. The return type is Dividends;
- GetDividends, whose return type is also Dividends, is used to obtain additional dividends from the height of a block.

SymbolList is a string list:

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

The type of Dividends is a map from token symbol to amount:

```
message Dividends {
    map<string, int64> value = 1;
}
```

20.10.2 Usage

ACS10 only unifies the standard interface of the dividend pool, which does not interact with the AElf chain.

20.10.3 Implementaion

With the Profit contract

A Profit Scheme can be created using the Profit contract's CreateScheme method:

```
State.ProfitContract.Value =
    Context.GetContractAddressByName(SmartContractConstants.ProfitContractSystemName);
var schemeToken = HashHelper.ComputeFrom(Context.Self);
State.ProfitContract.CreateScheme.Send(new CreateSchemeInput
{
    Manager = Context.Self,
    CanRemoveBeneficiaryDirectly = true,
    IsReleaseAllBalanceEveryTimeByDefault = true,
    Token = schemeToken
```

(continues on next page)

(continued from previous page)

```
});
State.ProfitSchemeId.Value = Context.GenerateId(State.ProfitContract.Value,␣
↪schemeToken);
```

The Context.GenerateId method is a common method used by the AElf to generate Id. We use the address of the Profit contract and the schemeToken provided to the Profit contract to calculate the Id of the scheme, and we set this id to State.ProfitSchemeId (SingletonState<Hash>).

After the establishment of the dividend scheme:

- ContributeProfits method of Profit can be used to implement the method Donate in ACS10.
- The Release in the ACS10 can be implemented using the method DistributeProfits in the Profit contract;
- Methods such as AddBeneficiary and RemoveBeneficiary can be used to manage the recipients and their weight.
- AddSubScheme, RemoveSubScheme and other methods can be used to manage the sub-dividend scheme and its weight;
- The SetSymbolList and GetSymbolList can be implemented by yourself. Just make sure the symbol list you set is used correctly in Donate and Release.
- GetUndistributedDividends returns the balance of the token whose symbol is includeded in symbol list.

With TokenHolder Contract

When initializing the contract, you should create a token holder dividend scheme using the CreateScheme in the TokenHolder contract(Token Holder Profit Scheme

```
State.TokenHolderContract.Value =
    Context.GetContractAddressByName(SmartContractConstants.
↪TokenHolderContractSystemName);
State.TokenHolderContract.CreateScheme.Send(new CreateTokenHolderProfitSchemeInput
{
    Symbol = Context.Variables.NativeSymbol,
    MinimumLockMinutes = input.MinimumLockMinutes
});
return new Empty();
```

In a token holder dividend scheme, a scheme is bound to its creator, so SchemeId is not necessary to compute (in fact, the scheme is created via the Profit contract).

Considering the GetDividends returns the dividend information according to the input height, so each Donate need update dividend information for each height . A Donate can be implemented as:

```
public override Empty Donate(DonateInput input)
{
    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {
        From = Context.Sender,
        Symbol = input.Symbol,
        Amount = input.Amount,
        To = Context.Self
    });
    State.TokenContract.Approve.Send(new ApproveInput
    {
        Symbol = input.Symbol,
        Amount = input.Amount,
```

(continues on next page)

(continued from previous page)

```
        Spender = State.TokenHolderContract.Value
    });
    State.TokenHolderContract.ContributeProfits.Send(new ContributeProfitsInput
    {
        SchemeManager = Context.Self,
        Symbol = input.Symbol,
        Amount = input.Amount
    });
    Context.Fire(new DonationReceived
    {
        From = Context.Sender,
        Symbol = input.Symbol,
        Amount = input.Amount,
        PoolContract = Context.Self
    });
    var currentReceivedDividends = State.ReceivedDividends[Context.CurrentHeight];
    if (currentReceivedDividends != null && currentReceivedDividends.Value.
↪ContainsKey(input.Symbol))
    {
        currentReceivedDividends.Value[input.Symbol] =
            currentReceivedDividends.Value[input.Symbol].Add(input.Amount);
    }
    else
    {
        currentReceivedDividends = new Dividends
        {
            Value =
            {
                {
                    input.Symbol, input.Amount
                }
            }
        };
    }
    State.ReceivedDividends[Context.CurrentHeight] = currentReceivedDividends;
    Context.LogDebug(() => string.Format("Contributed {0} {1}s to side chain␣
↪dividends pool.", input.Amount, input.Symbol));
    return new Empty();
}
```

The method Release directly sends the TokenHolder's method DistributeProfits transaction:

```
public override Empty Release(ReleaseInput input)
{
    State.TokenHolderContract.DistributeProfits.Send(new DistributeProfitsInput
    {
        SchemeManager = Context.Self
    });
    return new Empty();
}
```

In the TokenHolder contract, the default implementation is to release what token is received, so SetSymbolList does not need to be implemented, and GetSymbolList returns the symbol list recorded in dividend scheme:

```
public override Empty SetSymbolList(SymbolList input)
{
```

(continues on next page)

(continued from previous page)

```
    Assert(false, "Not support setting symbol list.");
    return new Empty();
}
public override SymbolList GetSymbolList(Empty input)
{
    return new SymbolList
    {
        Value =
        {
            GetDividendPoolScheme().ReceivedTokenSymbols
        }
    };
}
private Scheme GetDividendPoolScheme()
{
    if (State.DividendPoolSchemeId.Value == null)
    {
        var tokenHolderScheme = State.TokenHolderContract.GetScheme.Call(Context.
↪Self);
        State.DividendPoolSchemeId.Value = tokenHolderScheme.SchemeId;
    }
    return Context.Call<Scheme>(
        Context.GetContractAddressByName(SmartContractConstants.
↪ProfitContractSystemName),
        nameof(ProfitContractContainer.ProfitContractReferenceState.GetScheme),
        State.DividendPoolSchemeId.Value);
}
```

The implementation of GetUndistributdividendeds is the same as described in the previous section, and it returns the balance:

```
public override Dividends GetUndistributedDividends(Empty input)
{
    var scheme = GetDividendPoolScheme();
    return new Dividends
    {
        Value =
        {
            scheme.ReceivedTokenSymbols.Select(s => State.TokenContract.GetBalance.
↪Call(new GetBalanceInput
            {
                Owner = scheme.VirtualAddress,
                Symbol = s
            })).ToDictionary(b => b.Symbol, b => b.Balance)
        }
    };
}
```

In addition to the Profit and TokenHolder contracts, of course, you can also implement a dividend pool on your own contract.

20.10.4 Test

The dividend pool, for example, is tested in two ways with the token Holder contract.

One way is for the dividend pool to send Donate, Release and a series of query operations;

The other way is to use an account to lock up, and then take out dividends.

Define the required Stubs:

```
const long amount = 10_00000000;
var keyPair = SampleECKeyPairs.KeyPairs[0];
var address = Address.FromPublicKey(keyPair.PublicKey);
var acs10DemoContractStub =
    GetTester<ACS10DemoContractContainer.ACS10DemoContractStub>(DAppContractAddress,␣
↪keyPair);
var tokenContractStub =
    GetTester<TokenContractContainer.TokenContractStub>(TokenContractAddress,␣
↪keyPair);
var tokenHolderContractStub =
    GetTester<TokenHolderContractContainer.TokenHolderContractStub>
↪(TokenHolderContractAddress,
        keyPair);
```

Before proceeding, You should Approve the TokenHolder contract and the dividend pool contract.

```
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = TokenHolderContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
await tokenContractStub.Approve.SendAsync(new ApproveInput
{
    Spender = DAppContractAddress,
    Symbol = "ELF",
    Amount = long.MaxValue
});
```

Lock the position, at which point the account balance is reduced by 10 ELF:

```
await tokenHolderContractStub.RegisterForProfits.SendAsync(new RegisterForProfitsInput
{
    SchemeManager = DAppContractAddress,
    Amount = amount
});
```

Donate, at which point the account balance is reduced by another 10 ELF:

```
await acs10DemoContractStub.Donate.SendAsync(new DonateInput
{
    Symbol = "ELF",
    Amount = amount
});
```

At this point you can test the GetUndistributedDividends and GetDividends:

```
// Check undistributed dividends before releasing.
{
    var undistributedDividends =
        await acs10DemoContractStub.GetUndistributedDividends.CallAsync(new Empty());
    undistributedDividends.Value["ELF"].ShouldBe(amount);
}
var blockchainService = Application.ServiceProvider.GetRequiredService
↪<IBlockchainService>();
```

(continues on next page)

(continued from previous page)

```
var currentBlockHeight = (await blockchainService.GetChainAsync()).BestChainHeight;
var dividends =
    await acs10DemoContractStub.GetDividends.CallAsync(new Int64Value {Value =␣
↪currentBlockHeight});
dividends.Value["ELF"].ShouldBe(amount);
```

Release bonus, and test GetUndistributedDividends again:

```
await acs10DemoContractStub.Release.SendAsync(new ReleaseInput
{
    PeriodNumber = 1
});
// Check undistributed dividends after releasing.
{
    var undistributedDividends =
        await acs10DemoContractStub.GetUndistributedDividends.CallAsync(new Empty());
    undistributedDividends.Value["ELF"].ShouldBe(0);
}
```

Finally, let this account receive the dividend and then observe the change in its balance:

```
var balanceBeforeClaimForProfits = await tokenContractStub.GetBalance.CallAsync(new␣
↪GetBalanceInput
{
    Owner = address,
    Symbol = "ELF"
});
await tokenHolderContractStub.ClaimProfits.SendAsync(new ClaimProfitsInput
{
    SchemeManager = DAppContractAddress,
    Beneficiary = address
});
var balanceAfterClaimForProfits = await tokenContractStub.GetBalance.CallAsync(new␣
↪GetBalanceInput
{
    Owner = address,
    Symbol = "ELF"
});
balanceAfterClaimForProfits.Balance.ShouldBe(balanceBeforeClaimForProfits.Balance +␣
↪amount);
```

20.10.5 Example

The dividend pool of the main chain and the side chain is built by implementing ACS10.

The dividend pool provided by the Treasury contract implementing ACS10 is on the main chain.

The dividend pool provided by the ACS10 contract implementing ACS10 is on the side chain.

CHAPTER 21

Command line interface

21.1 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.

21.1.1 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.

21.1.2 Install aelf-command

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

21.1.3 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 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 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
  -a, --account <account>                                The address of AElf wallet
```

(continues on next page)

(continued from previous page)

```
  -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]

Open a node REPL
```

(continues on next page)

(continued from previous page)

```

Options:
  -h, --help  output usage information

Examples:

aelf-command console
...
```

21.2 Commands

21.2.1 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.

21.2.2 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

Examples:
```

(continues on next page)

(continued from previous page)

```
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
```

21.2.3 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
...
```

21.2.4 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
```

21.2.5 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":
↪"AAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪",
  "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*.

21.2.6 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 need to wait for the organization members to approve your proposal and you can release your proposal by calling `releaseApprove` and `releaseCodeCheck` in this order.

21.2.7 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;

21.2.8 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"}'
```

21.2.9 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"}'
```

21.2.10 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",
```

(continues on next page)

(continued from previous page)

```
  "LastIrreversibleBlockHash":
↪"497c24ff443f5cbd33da24a430f5c6c5e0be2f31651bd89f4ddf2790bcbb1906",
  "LastIrreversibleBlockHeight": 4229063,
  "BestChainHash": "59937e3c16860dedf0c80955f4995a5604ca43ccf39cd52f936fb4e5a5954445",
  "BestChainHeight": 4229086
}
```

21.2.11 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:

↪'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
↪',
  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 }
```

21.2.12 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
```

21.2.13 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+DggIEv
↪AkWcIrCxvCX/
↪Te3fGHVXFxE8xsnfT1HtMgwIoJro6AUQjOa1pQE4TkqCATA0MjUzZGM1MTMwZmE0ZjVjYmFhYmZiYmFiYmFhZDVmMTMwNDMwMTI
↪836zXZTbntbqyjgtoBHAEegwIoJro6AUQzOybpgF6DAigmujoBRCk8MG1AnoLCKGa6OgFEOCvuBF6CwihmujoBRCg/
↪JhzegwIoZro6AUQ9Lml1wF6DAihmujoBRDYyOO7AnoMCKGa6OgFEKy+ip8DkAEOEp8CCoIBMDQ4MWMyOWZmYzVlZjI5NjdlMjVi
↪", "" ]',
     Height: 123,
     Time: '2019-07-01T13:39:45.8704899Z',
     ChainId: 'AELF',
     Bloom:

↪'0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪',
     SignerPubkey:

↪'04253dc5130fa4f5cbaabfbbabbaad5f1304301232ad6316d72bceb0506ac0387ec180256431e9dd2e44199a9129320cdc
↪' },
  Body:
   { TransactionsCount: 1,
     Transactions:
      [ 'a365a682caf3b586cbd167b81b167979057246a726c7282530554984ec042625' ] } }
```

```
aelf-command get-blk-info␣
↪ca61c7c8f5fc1bc8af0536bc9b51c61a94f39641a93a748e72802b3678fea4a9 true
```

21.2.14 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
```

21.2.15 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 22

Wallet and Block Explorer

22.1 Explorer

Github

Currently, the explorer provides functions such as viewing blocks, transactions, purchasing resources, voting and node campaigning as well as viewing contracts.

22.2 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.

22.3 Web Wallet

Github

The Web Wallet provides basic transaction related functionality.

22.3.1 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 23

aelf-web-extension

You can get more information in Github

23.1 For User

release version, please waiting

dev version

If you are using qq browser,etc, you can add the extention too.

23.1.1 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.

23.2 For Dapp Developers

23.2.1 Interaction Flow

- 1.Make sure the user get the Extension
- 2.Connect Chain
- 3.Initialize Contract
- 4.Call contract methods

23.2.2 How to use

If you need complete data structure. you can *click here*

- *0. Check Extension Demo*
- *1. GET_CHAIN_STATUS*
- *2. CALL_AELF_CHAIN*
- *3. LOGIN*
- *4. INIT_AELF_CONTRACT*
- *5. CALL_AELF_CONTRACT / CALL_AELF_CONTRACT_READONLY*
- *6. CHECK_PERMISSION*
- *7. SET_CONTRACT_PERMISSION*
- *8. REMOVE_CONTRACT_PERMISSION*
- *9. REMOVE_METHODS_WHITELIST*

23.3 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'
                    }
                }
            }
        ]
    }
}
```

23.3.1 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
});
```

23.3.2 1.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"
// }
```

23.3.3 2.CALL_AELF_CHAIN

```
const txid = 'c45edfcca86f4f528cd8e30634fa4ac53801aae05365cfefc3bfe9b652fe5768';
aelf.chain.getTxResult(txid, (err, result) => {
    console.log('>>>>>>>>>>>>> getTxResult >>>>>>>>>>>>>');
    console.log(err, result);
});

// result = {
//     Status: "NotExisted"
//     TransactionId:
↪"ff5bcd126f9b7f22bbfd0816324390776f10ccb3fe0690efc84c5fcf6bdd3fc6"
// }
```

23.3.4 3. 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'
//     }]
// }
```

23.3.5 4.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 (),
//     ....
// }
```

23.3.6 5.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
// }
```

23.3.7 6.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'
//          }
//      ]
//  }
```

23.3.8 7.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'
//      }]
// }
```

23.3.9 8.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'
//     }]
// }
```

23.3.10 9.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'
//      }]
// }
```

23.4 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.
```

23.5 Project Information

We use ECDH to use public key to encryt data and private key to decrypt data.

CHAPTER 24

DevOps

24.1 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.

24.2 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
```

24.3 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.

24.3.1 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.

24.3.2 Perf testing

The performance testing is crucial to AElf since a strong point of our system is speed.

24.4 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 25

QuickStart

25.1 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.

25.1.1 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*.

25.1.2 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.

25.1.3 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/netcoreapp3.1/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
```

25.1.4 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)
```

25.1.5 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 26

todo

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:

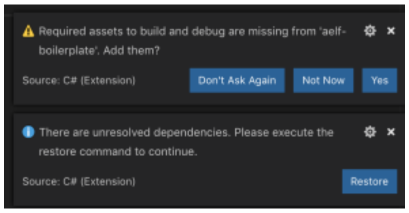
```
git clone https://github.com/AElfProject/aelf-boilerplate
```

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**.


Required assets to build and debug are missing from 'aelf-boilerplate'. Add them?

Source: C# (Extension)

There are unresolved dependencies. Please execute the restore command to continue.

Source: C# (Extension)

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/netcoreapp3.1/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;       // the address of the signer
    Address to;         // the address of the target contract
    string method_name; // the method to execute
    bytes params;       // the parameters to pass to the method
    bytes signature;    // 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();
}
```

Next

The execution context also exposes functionality for sending inline transactions; the next article will give you more details on how to generate inline calls.

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
{
```

(continues on next page)

(continued from previous page)

```
    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.
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**).