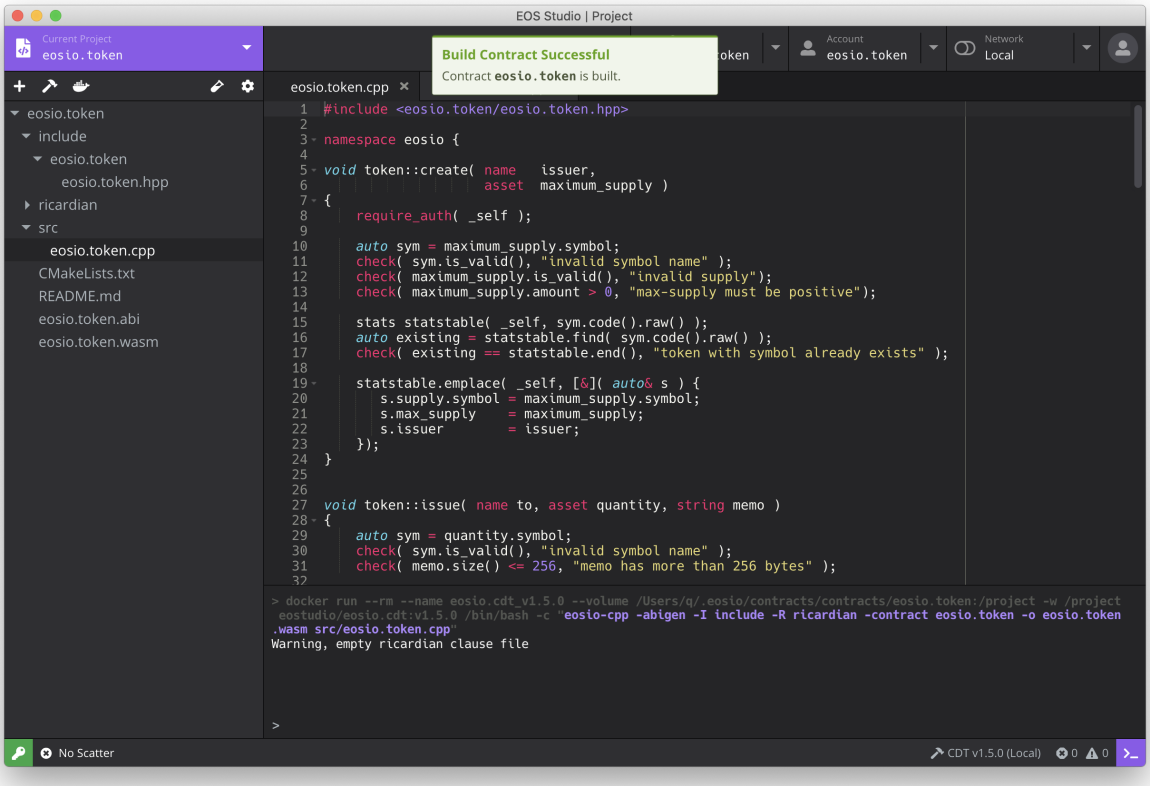
EOS Studio Documentation

Phil Li

Nov 19, 2019

INTRODUCTION



EOS Studio (https://www.eosstudio.io) is a graphical IDE for EOSIO dApp development. It was first launched in Febuary 2019 and quickly became the most popular tool for EOSIO, attracting thousands of EOSIO developers worldwide. At present, EOS Studio comes in two versions: a *desktop version* that supports Mac OS, Windows and Linux operating systems, as well as a *web version* that allows dApp development in a browser.

By integrating various tools required for EOSIO in a unified application, EOS Studio aims to provide a powerful and easy-to-use environment for dApp development. With EOS Studio, developers can complete the entire dApp development process in a single application. The key features of EOS Studio include

- C++ code editor with EOSIO syntax highlighting
- Built-in EOSIO.CDT and *Cloud CDT*
- Interactive *Contract Inspector*
- Version manager for EOSIO software

This documentation will introduce how to use EOS Studio for dapp development, explain the functions of each module of EOS Studio, and also present some high-quality EOSIO smart contracts.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Getting Started

EOS Studio is available in two versions. Developers are welcome to choose either one that better fits their purpose in development.

- *EOS Studio Desktop* is a stand-alone desktop appliction that supports Mac OS, Windows and Linux operating systems. It will also help you install and manage other tools required in the development, including EOSIO and EOSIO.CDT.
- *EOS Studio Web* is a complete IDE that runs entirely in the browser. Developers can open a link and start dApp development immediately without any pre-installation. Meanwhile, the *Cloud CDT* and cloud-based network ensure that EOS Studio Web can provide complete dApp development capabilities.

In the following sections, we will demonstrate how to use both versions to create, build, deploy, and execute a smart contract. You will be able to learn their difference and choose the appropriate one for your purposes.

1.1.1 EOS Studio Desktop

EOS Studio Desktop is a stand-alone desktop appliction, supporting Mac OS, Windows and Linux.

Download

You can download the installation package from the following links:

- **Mac OS**: https://download.eosstudio.io/mac
- **Windows**: https://download.eosstudio.io/win
- **Linux**: https://download.eosstudio.io/linux

Set up the environment

When you launch EOS Studio for the first time, there will be a welcome screen to help you set up the tools required for EOSIO dApp development. That includes:

- EOSIO is the main software to run a EOSIO-based blockchain. It includes
  - `nodeos`: the core executable that runs the EOSIO blockchain for block production and providing API endpoints. You need to start one for local development;
  - `cleos`: a command line tool to query the EOSIO blockchain;
  - `keosd`: a commane line wallet to manage keypairs and sign transactions;

- EOSIO.CDT which stands for *Contract Development Toolkit* is used to compile C++ source codes to WebAssembly, a binary format EOSIO uses to run smart contracts.

EOS Studio Desktop uses Docker to install and run the above tools. With dockerized EOSIO and EOSIO.CDT, it's easier to work accross differenct operating systems. If you don't have Docker yet, the welcome page will guide you to install it.

In EOS Studio, `cleos` and `keosd` are not necessary.

Note: **[Windows]** There are two types of Docker: Docker Desktop (for Window 10 Pro only) and Docker Toolbox (for all the others). Be sure to know which type of Docker your are using. EOS Studio works better with Docker Desktop, but it has some compatible issues with Docker Toolbox. In that case, we recommend to use *Cloud CDT* and Cloud-hosted Network.

Note: **[Linux]** After install docker, you also need to allow non-privileged users to run Docker commands. See instructions here.

After Docker is installed and launched, the welcome page will further assist you to download docker images for EOSIO (eostudio/eos) and EOSIO.CDT (eostudio/eosio.cdt). Both of them have many versions, but in most cases you only need to install the latest. If you have previous projects that only work with older EOSIO or EOSIO.CDT, you can download multiple versions and EOS Studio will help you to manage them.

Warning: Do not use the docker image on mainnet or as a block producer. They are made for development purpose only.

Create a new project

Once you finish the installations, EOS Studio will go to the page of your project list. It is empty now, so let's click the *Create* button and create a new project.

The new project will be initilized with some basic codes for a smart contract. You can now press the **:fa:'gavel'** *hammer* button in the *toolbar* to build the project. This will run the EOSIO.CDT docker image to compile the contract and export a `.wasm` file and an `.abi` file.

- The `wasm` file is a WebAssembly binary that will run on the EOSIO blockchain
- The `abi` file is a json object that defines the contract actions and data tables with type information of action parameters and table rows.

Start a local network

Before going forward, you need to start a local network. Switch to the Network Page where you can see all installed EOISO softwares. Click the *Run* button to start a network. EOS Studio will

Create an account

Deploy the contract

> **Warning:** Be careful that do not deploy to `eosio` account, unless you know what you are doing.

Execute the contract action

1.1.2 EOS Studio Web

EOS Studio Web is available at https://app.eosstudio.io

Log in with GitHub

You can look at other's projects. To create your own project, you need to login first.

Create a new project

press the button and create the project

Cloud CDT

use cdt to build the project, generating `.wasm` and `.abi` files

Cloud-hosted Network

Create an account

deploy the contract to your account

Run your first smart contract

In addition, many blockchain teams are sharing their open source smart contracts on EOS Studio Web to help new users get started.

The main featurs of EOS Studio Web include:

- An online EOSIO code editor that supports syntax highlight, auto-complete and inline notification of build errors
- A cloud-based EOSIO.CDT smart contract compiler with the option to choose versions from v1.3 to v1.6
-

1.2 EOSIO Fundamentals

This chapter will introduce some basic concepts about EOSIO or blockchain based applications.

1.2.1 Accounts

Blockchains like Bitcoin or Ethereum use *addresses* to represent individuals in transactions. Tokens are transfered from one address to another, and each address has its own token balance. However, EOSIO-based blockchain use *accounts* as the basic unit to store tokens and act as individuals in blockchain transactions.

Account Name

An EOSIO account can have a human readable name so it will be easy to remember. The account name is a string of max length 12 consists of small letters a-z, digits 1-5 and dot.

Create an Account

A new EOSIO account needs to be created by another existing account. Creating an account will require some EOS tokens to purchase the *resources* needed to store the account's basic information, such as its own name and token balance.

Permissions

A special feature EOSIO offers is that an account can process multiple levels of permissions, each of which has unequal authorities to approve different sets of transactions. Such design can provide greater security for EOSIO blockchains because users can set up and use a low-privilege permission in daily transactions. Losing this permission wouldn't cause too much damage because it can only perform limited transactions, and users can use a high-privilege permission to recover the lost one.

1.2.2 Resources

The EOSIO blockchain defines three types of resources to compensate the usage of the network:

- CPU for computation time
- NET for network bandwidth
- RAM for data storage

CPU – Computation time

Any transaction posted to the blockchain network, either a token transfer or an execution of smart contract actions, needs to consume some CPU time to be properly processed and packed into blocks. The total amount of CPU time is limited by the harware, so a certain amount of EOS needs to be *staked* in exchange for CPU usage rights. The network will prorate the CPU time that each account can use based on the total amount of CPUs collateralized on all accounts. This time represents the total CPU length that can be used in 24 hours.

Staked EOS in exchange for CPU can be refunded if some computation time will not be used in the future.

so that transactions tenically free. It takes 72 hours.

NET – Network bandwidth

NET's distribution mechanism is the same as the CPU, but also through the mortgage method. To allocate the use of network bandwidth.

RAM – Memory usage

The total amount of storage space on the chain is limited, so storing data requires buying RAM. RAM needs to be purchased and the price is automatically adjusted by the bancor algorithm. The more people you buy, the higher the RAM price, and vice versa.

REX

If an account need to use a large amount of CPU or NET for a short period of time, it needs to stake a huge amount of EOS token.

1.2.3 Decentralized Application

EOSIO blockchain use account system to

Kickoff

You are three easy steps away from being a blockchain developer:

1. EOS Studio IDE and EOS account preparation
   - Launch EOS Studio IDE Web: https://app.eosstudio.io
   - Login into Studio using Github account
   - Create an EOS account in Cloud network
2. Backend preparation
   - Create a project
   - Build
   - Deploy
3. Frontend preparation (TODO add a simple test page with EOSJS API call)
   - Launch test page
   - Fill the account
   - Click 'Hi' button

That's it, that's all, you are a blockchain developer now.

Talk is cheap. Show me the code.

Backend (Smart Contract)

- using C++
- Header file (hpp) and source file (cpp)

hpp:

- [[eosio::contract]] syntax
- Constructor and member initializer lists (https://en.cppreference.com/w/cpp/language/initializer_list)
- Actions

- Define table and table instance

cpp:

- function implementation
- require_auth (maybe ignore it)
- multi_index get a record (get)
- multi_index add a new record (emplace)
- multi_index replace an existing record (modify)
- C++ callback function

Frontend (TODO)

From 'Hello world!' to 'Hello xxx!' (xxx from blockchain) after clicking a button.

Offer a reset button.

- React setup
- EOSJS setup
- onClick binding
- EOSJS API fetch
- update state

1.2.4 Smart Contract

Smart contracts are similar to backend servers but they are running on blockchains. Users will interact with smart contracts through their APIs called *actions*, and the persisted data is stored in

Actions

operations

Tables

data

`scope`

1.3 Overview

EOS Studio comes with a simple and intuitive layout. The UI is divided into four pages, switchable through the navbar buttons.

- *Project Editor* is the main interface to display your EOSIO project and provide a EOSIO-tailored editor to code, build and deploy your smart contracts.
- *Contract Inspector* is a convenient tool to debug smart contracts. It allows you to easily execute contract actions and visualize the table data.

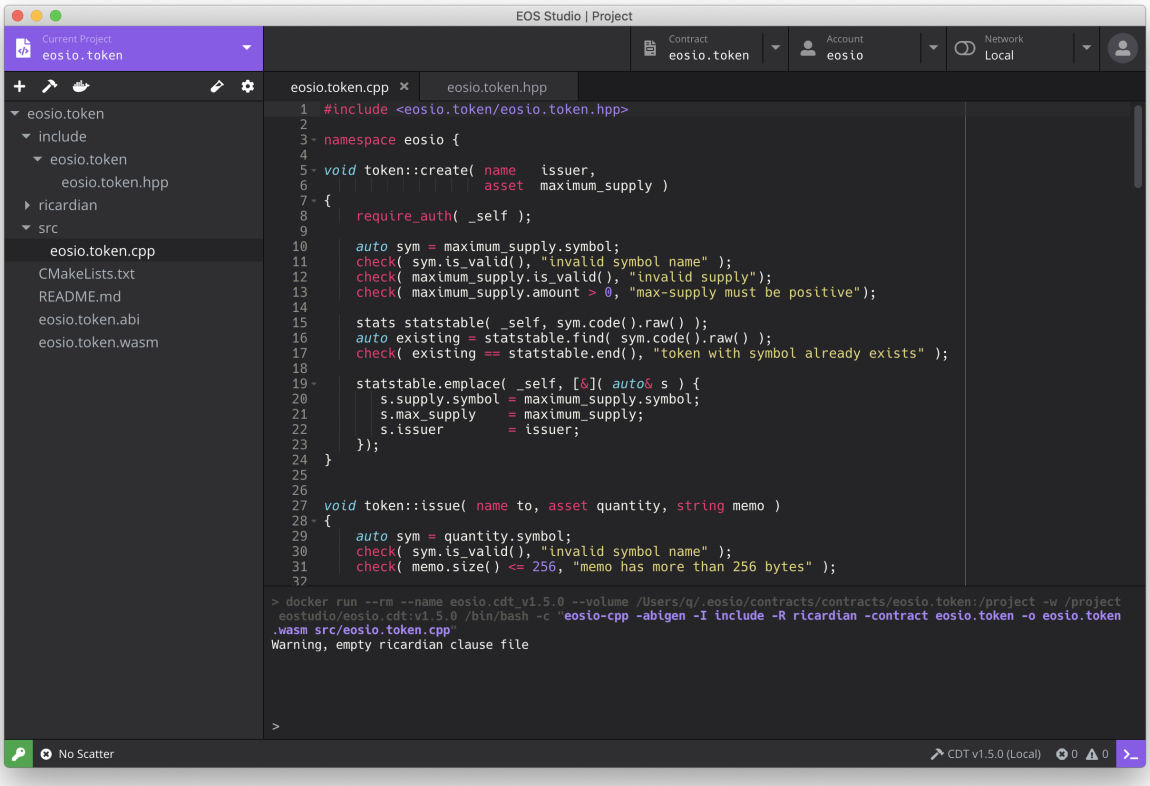
- *Account Viewer* is a page to view account information and perform account related operations.
- *Network Manager* can help you switching between local network, different testnets, and EOS Mainnet, as well as showing the information of the selected network. You can also connect to a custom network by API endpoint. For local network, it also integrates the *EOSIO Version Manager* to install and manage multiple versions of the EOSIO software.

At the bottom, the *Bottom Bar* will provide easy access to other tools that might be useful during the development. Tools listed on the left side are usually needed in many places, so they are fixed in the Bottom Bar.

- The *Keypair Manager* can help you to create, import or export keypairs. Be aware that EOS Studio will not encrypt the private keys and don't use any of those in the mainnet.
- The *Scatter* button will show connection status to Scatter Desktop. EOS Studio will use Scatter to sign transactions when you are working on the mainnet.

However, tools on the right will change according to the current active page and different pages have their own available tools. The following sections will introduce them by pages respectively.

1.4 Project Editor



1.4.1 Main Components

Code Editor

eosio.token.cpp | eosio.token.hpp

```cpp
#include <eosio.token/eosio.token.hpp>

namespace eosio {

void token::create( name   issuer,
                    asset  maximum_supply )
{
    require_auth( _self );

    auto sym = maximum_supply.symbol;
    check( sym.is_valid(), "invalid symbol name" );
    check( maximum_supply.is_valid(), "invalid supply");
    check( maximum_supply.amount > 0, "max-supply must be positive");

    stats statstable( _self, sym.code().raw() );
    auto existing = statstable.find( sym.code().raw() );
    check( existing == statstable.end(), "token with symbol already exists" );

    statstable.emplace( _self, [&]( auto& s ) {
       s.supply.symbol = maximum_supply.symbol;
       s.max_supply    = maximum_supply;
       s.issuer        = issuer;
    });
}


void token::issue( name to, asset quantity, string memo )
{
    auto sym = quantity.symbol;
    check( sym.is_valid(), "invalid symbol name" );
    check( memo.size() <= 256, "memo has more than 256 bytes" );
```

The code editor has integrated some of the most practical tools for contract development. For example, it supports highlight and autocompletes for EOSIO-specific syntax.

The code editor will render markdown files.

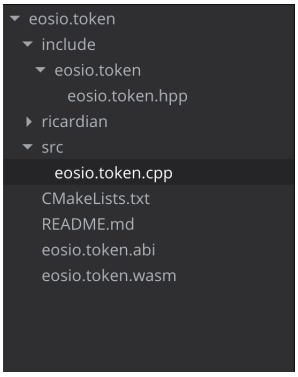
The `README.md` file will serve as the main page for a project.

Toolbar

In the toolbar menu at the top left, there are some handy buttons that help you easily do common operations such as

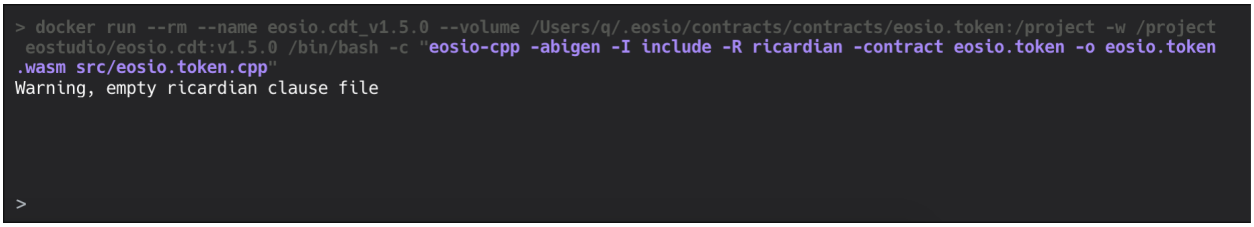
- New Contract:
- Build: use EOSIO.CDT to compile smart contract
- Deploy: deploy the `wasm` and `abi` files to a specific account
- Test (only for desktop): run test cases; will initialize the test framework when the button is pressed for the first time
- Project Settings: open the project settings page

File Tree



In the file tree, you can xxx the project files.

Terminal



The terminal is mainly used to display EOSIO.CDT outputs and test outputs.

1.4.2 Types of Projects

Local Project (only for desktop)

A local project is saved on your disk.

Remote Project

A remote project is saved on EOS Studio's cloud service. You can use both EOS Studio Desktop and EOS Studio Web to access a remote project.

Others' Shared Project

Open projects shared by others

Code editor

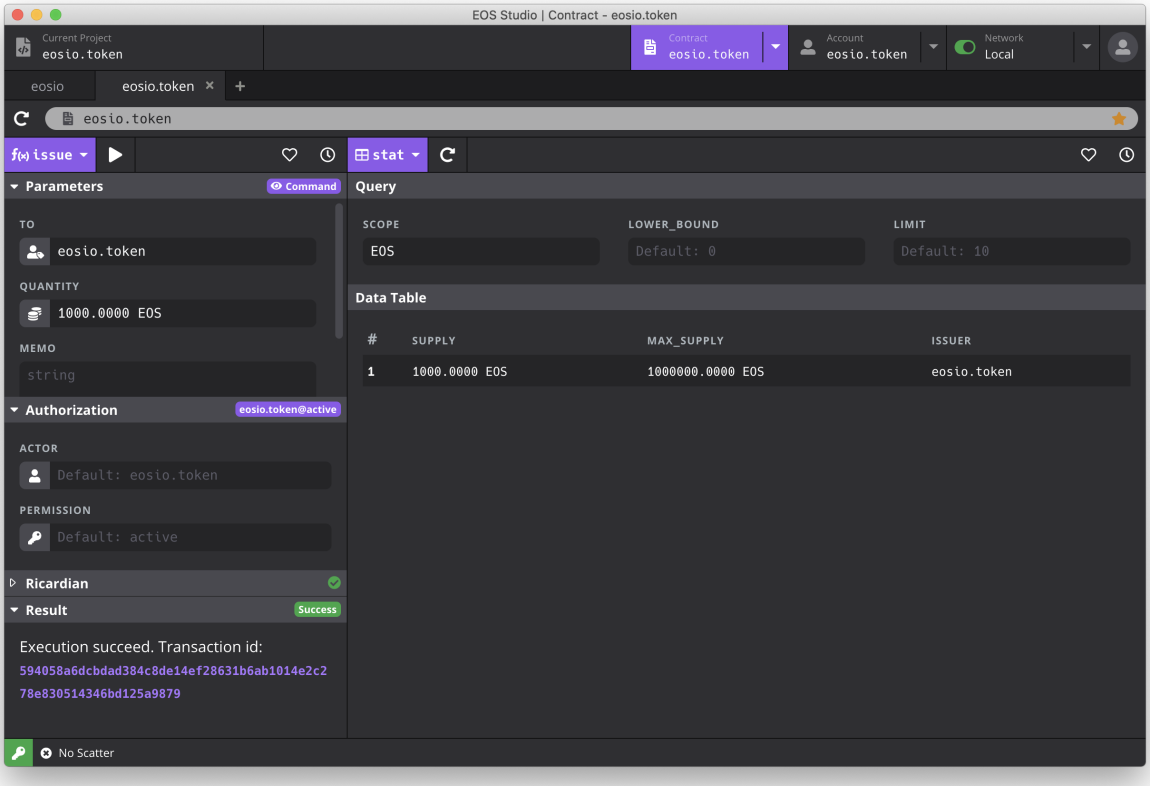
1.4.3 Project Settings

EOS Studio provides a Project Settings page to easily view and modify the .eosproj file, accessible by clicking the cog button in the toolbar menu. The first item defines the main file for the project and the compile process will start from this file. The second contract name item corresponds to the –contract attribute for CDT command line and is

also required for compilation. The following items are optional and usually used for some advanced configuration in compilation. You can refer the CDT command line documentation to learn how to use them.

In the below Deployment Settings, you can specify the account to which you want to deploy your smart contract. EOS Studio supports local, Kylin and Jungle testnets, and EOSIO mainnet, so you can specify them separately. For example, if we enter newcontract in the local config line, you will see the name also appears next to the deploy button in the toolbar. Now if we click the button, EOS Studio will deploy the latest compiled codes to the newcontract account.

1.5 Contract Inspector



The Contract Page provides the necessary tools to inspect and debug smart contracts. In order to view multiple contracts at the same time, EOS Studio uses tabs to support for opening multiple contracts. You can click on the tab to quickly switch the contract you want to view. At the same time, the commonly used contract Account can be starred.

Just below the tabs, there is an address bar where lets you enter the contract account name. EOS Studio will automatically read the abi file in the account to check the contract based on the contract account you entered.

The EOS Studio Contract Inspector has two parts:

1) a panel to execute actions on the left, and

2) a panel to query table data on the right.

In the dropdown menus at the top left for each panel, you can easily view all the actions and all the tables respectively.

If a smart contract is found in the account, EOS Studio will parse the abi file to visualize its actions and tables.

1.5.1 Actions

Actions are shown on the right. You can switch the action you want to call through the dropdown menu.

Form for Input Parameter

A form for inputs will be generated from the abi to make it easier to enter parameters.

The input of the action contains many types, and EOS Studio will process the input parameters according to the type:

- For type of `uint64_t`, `uint32_t`,
- For type of `permission`,

You can view the raw transaction command by clicking the *View Command* button. It will tell you what command, including the authorization set below, EOS Studio is going to execute when you press the *Run* button. It will show you both the `cleos` command and the `eosjs` script.

Authorization

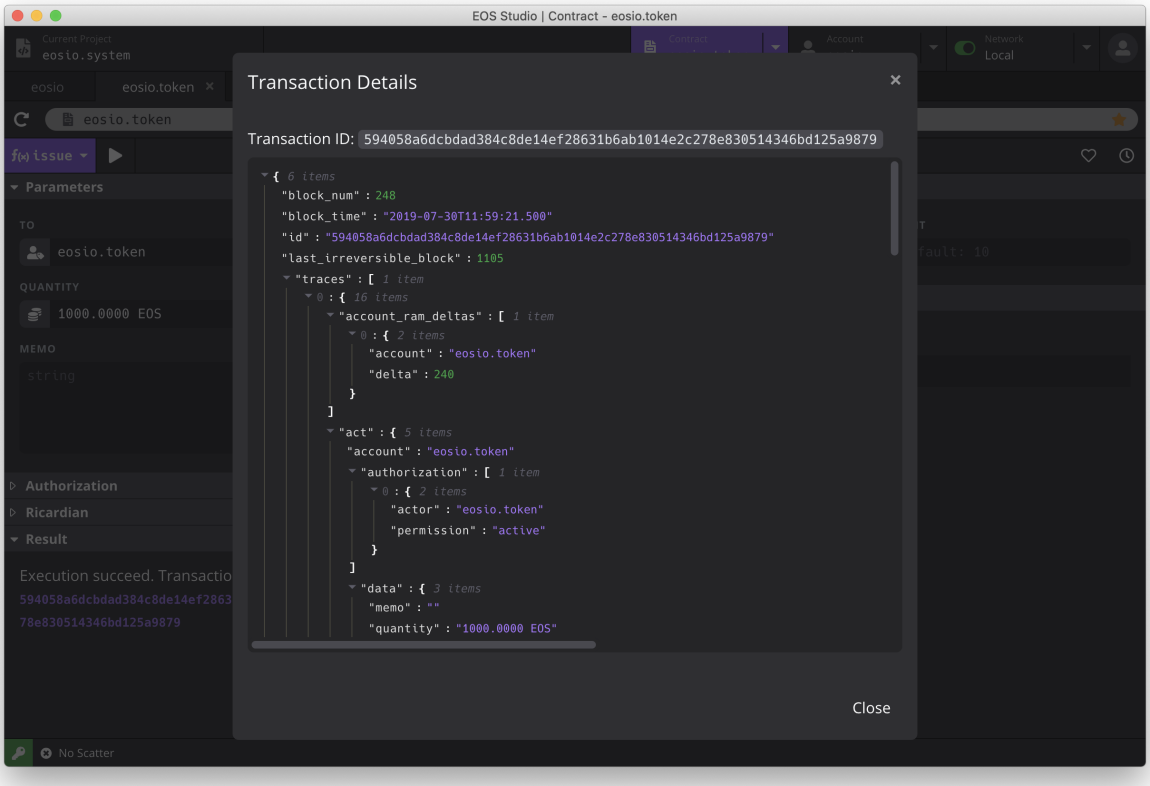
You can change the `actor` and `permission` used to sign the transaction. By default, EOS Studio will use `{account}@active` which `account` is the current selected account.

EOS Studio doesn't support multisig yet.

Ricardian

Result

The result after calling the contract will be displayed here. If the result is, you will see the transaction hash, click to see the complete transaction details.



If the transaction fails, you can see the error message.

Execution History and Bookmarks

When we are debugging a smart contract, we often need to call the same actions repeatedly, and constantly refresh the table to view the most recent data. Most HTTP API clients will store call history and have bookmarks to save common-used parameters. EOS Studio has these features too. Within the clock icon buttons on the top right for each panel, you can see the histories for action executions and table queries. They would be convenient if you need to check past execution results, or simply want to re-run with the same previous parameters.

EOS Studio can also save frequently used parameters to bookmarks. For example, if I want to issue 10 EOS to myself repeatedly, I can save it so I don't need to enter them again. Go to the heart icon and select add to bookmarks, you will see the contract action, the authorizer, and parameters to execution with. Just enter a name and save it, and you can access it in bookmarks anytime in the future.

The record of the calling contract (including parameters and results) will be saved in the history for easy query. In addition, you can add common used parameters to *bookmarks*.

1.5.2 Tables

Tables are shown on the right. You can switch the table you want to view through the dropdown menu.

1.6 Account Viewer

The Account Page help you to check basic account information as well as perform some account operations. Similar to the Contract Page, the Account Page is also organized using tabs to allow multiple accounts being open at the same time. You will also using the address bar below the tabs to enter the account name. EOS Studio will read the account information and display it in the page below.

The Contract and Account pages share the same starred account list. You can also use the star to mark the commonly used account.

1.6.1 Basic Information

Display basic user information, including token balance, resources (CPU/NET/RAM), and permission keys.

When you are using a local network and the balance is always zero even though you have issued or transfered some tokens to the account, that is probably because you didn't setup the core symbol for the network. See here for instructions.

1.6.2 Transaction History

You can see the transaction history below the basic information.

1.6.3 Create a New Account

To create a new account, click the xxx and enter the account name. You also need to select a public key from the Keypair Manager to use as `owner` and `active` keys. If there is no key yet, you need to open the Keypair Manager and create one first. EOS Studio will check whether the account name has been used or not. Once a new account is created, it will be starred automatically.

If you want to import a created account, just type the account name in the address bar and *star* the account. However, you may not have the permission to operate this account. You can go to the Keypair Manager and import its private keys.

1.6.4 Tools

EOS Studio provides a handy tool for some common account operations. These tools can be passed to the right of the address bar button to access.

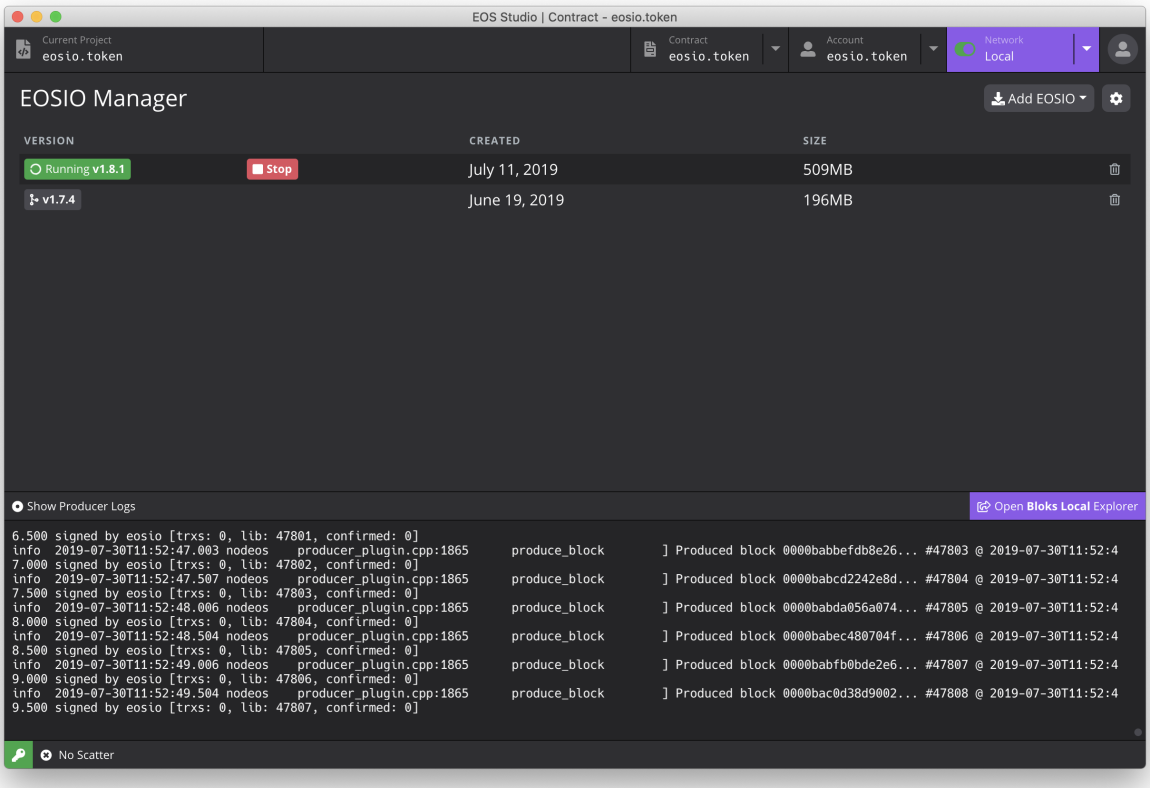
- **Transfer**: make a transfer of the core symbol tokens.
- **Set the** `eosio.code` **permission** - xxxx actions need `eosio.code` permission to run
- **Faucet (testnets only)** - click the button to claim some free tokens on a testnet
- **Buy RAMs** - buy `100 KB` RAMs.

1.7 Network Manager

Network are use to switch connected EOSIO network.

1.7.1 Local Network

(Only for EOS Studio Desktop)



EOSIO Version Manager

A table of installed EOSIO versions is listed here. If you want to install another one, click the *install* button and select the version you want to install. You can also delete unwanted versions.

To start a local network, select the version you want to start and click the *Run* button. EOS Studio will start a docker container and assemble the command to run `nodeos`. Once it is started, you can see the block producing logs in the log terminal below.

Advanced Configuration

EOS Studio allow you to modify the paramters to run `nodeos`. Click the *cog* button to open advanced configuration window. Here you will see a list of configurations, and please check `nodeos` documentation to understand how to use them.

Logs of Block Production

You can toggle the button and hide . . .

1.7.2 Cloud-hosted Network

This is a for-development testnet provided by dfuse.

1.7.3 Remote Networks

Other networks EOS Studio supports

- The EOSIO Mainnet
- Jungle 2.0 testnet
- CryptoKylin testnet

You can also connect to a custom networks

Basic Information

API Endpoints and Chian ID

Access to Block Explorers

Blocks

1.8 Bottom Bar

1.8.1 Keypair Manager

Keypair Manager used to manage keypairs.

1.8.2 Scatter

1.9 eosio.token

1.9.1 Introduction

The `eosio.token` contract defines the structures and actions that allow users to create, issue, and manage tokens on EOSIO based blockchains. The core token `EOS` of the EOSIO mainnet are issued under the account `eosio.token` using this smart contract.

- GitHub repo: https://github.com/EOSIO/eosio.contracts/tree/master/contracts/eosio.token
- EOS Studio: https://app.eosstudio.io/eosio/eosio.token

1.9.2 Types

There are a few types used in `eosio.token` as basic data structures. You can click the link in the action definitions to see how the types are defined.

1.9.3 Smart Contract

Actions

class token

ACTION **create** (eosio::*name issuer*, eosio::*asset maximum_supply*)
Create a token in supply of `maximum_supply` with an `issuer` account. If successful, a new entry in *`stat`* table for token symbol scope will be created. Transaction must be signed by the contract account itself.

ACTION **issue** (eosio::*name to*, eosio::*asset quantity*, string *memo*)
Issue `quantity` of tokens to account `to` with an optional `memo` that accompanies the token issue transaction. The token needs to be created in advance. Transaction must be signed by the `issuer`.

ACTION **transfer** (eosio::*name from*, eosio::*name to*, eosio::*asset quantity*, string *memo*)
Transfer `quantity` of tokens from account `from` to account `to`, with an optional `memo` that accompanies the transfer transaction. The token needs to be created in advance. Transaction must be signed by account `from`.

ACTION **open** (eosio::*name owner*, eosio::*symbol symbol*, eosio::*name ram_payer*)
Allows `ram_payer` to create an account `owner` with zero balance for token `symbol` at the expense of `ram_payer`. Transaction must be signed by account `ram_payer`.

ACTION **close** (eosio::*name owner*, eosio::*symbol symbol*)
This action is the opposite for *`open()`*, it closes the account `owner` for token `symbol`.

ACTION **retire** (eosio::*asset quantity*, string *memo*)
The opposite of *`create()`*. If all validations succeed, it debits the statstable.supply amount.

Tables

class token

TABLE **stat**

```
// scope is token symbol
eosio::asset supply; // supply.symbol is the primary key
eosio::asset max_supply;
eosio::name issuer;
```

TABLE **accounts**

```
// scope is owner
eosio::asset balance; // balance.symbol is the primary key
```

1.10 EOSIO.CDT

1.10.1 Header Files

symbol.hpp

```
#include <eosio/symbol.hpp>
```

class eosio::**symbol_code**

Information about a token symbol, the symbol can be up to 7 characters long.

Example:

```
auto symbol_code = eosio::symbol_code("EOS");
```

symbol_code(std::string_view *str*)

Construct a new `symbol_code` initialising value with `str`

symbol_code(uint64_t *raw*)

Construct a new `symbol_code` initialising value with `raw`

std::string **to_string**()

Returns the symbol name as a string

uint64_t **raw**()

Returns the raw `uint64_t` value for the symbol

friend bool **operator==**(**const** *symbol_code* &*a*, **const** *symbol_code* &*b*)

friend bool **operator!=**(**const** *symbol_code* &*a*, **const** *symbol_code* &*b*)

friend bool **operator<**(**const** *symbol_code* &*a*, **const** *symbol_code* &*b*)

private uint64_t **value**

Stores the symbol code as a `uint64_t` value

class eosio::**symbol**

Used to define a token's *symbol_code* and `precision` (digits after the decimal).

Example:

```
auto symbol = eosio::asset("10.0000 EOS").symbol;
symbol.code(); // eosio::symbol_code("EOS")
symbol.precision(); // 4
```

For example, `10.0000 EOS` has *symbol_code* **EOS** and *precision* **4**. A `symbol` can be written as `{precision},{symbol}` (in above example, `4,EOS`).

symbol(eosio::*symbol_code* *sc*, uint8_t *precision*)

symbol(std::string_view *sc*, uint8_t *precision*)

eosio::*symbol_code* **code**()

uint8_t **precision**()

class eosio::**extended_symbol**

A type of token is created by the *token::create()* action. The same contract account cannot `create` two types of tokens with the same `symbol`, but two different accounts deployed with the same `eosio.token` contract can create separate tokens with identical `symbol`.

To prevent such vulnerability, a `extended_symbol` s could be equal but represent two different tokens.

int64_t **amount** = 0

eosio::*symbol* **symbol**

asset.hpp

```
#include <eosio/asset.hpp>
```

class eosio::**asset**

Used to specify some amount of tokens. It consists of an `amount` property and a `symbol` property. For example, `10.0000 EOS` is an `asset` with `amount` equals `10 * 10^4` and `symbol` equals `4,EOS`.

name.hpp

```
#include <eosio/name.hpp>
```

struct eosio::**name**

Mainly used to represent an EOSIO account name. Name string can only have small letters a-z, digits 1-5 or dot, and max 12 characters. The name is saved as a `uint64_t`.

name(std::string_view *str*)

Construct a new `name` initialising value with `str`

name(uint64_t *raw*)

Construct a new `name` initialising value with `raw`

std::string **to_string**()

Returns the name as a string

uint64_t **raw**()

Returns the raw `uint64_t` value for the name

friend bool **operator==**(**const** *name* &*a*, **const** *name* &*b*)

friend bool **operator!=**(**const** *name* &*a*, **const** *name* &*b*)

friend bool **operator<**(**const** *name* &*a*, **const** *name* &*b*)

private uint64_t **value**

Stores the name as a `uint64_t` value

time.hpp

```
#include <eosio/time.hpp>
```

class eosio::**microseconds**

Microseconds.

microseconds(int64_t *count* = 0)

static *microseconds* **maximum**()

Maximum `0x7fffffffffffffll`

int64_t **count**()

int64_t **to_seconds**()

int64_t **_count**

The value used in serialization

inline *microseconds* eosio::**milliseconds**(int64_t *ms*)

inline *microseconds* eosio::**seconds**(int64_t *s*)

inline *microseconds* eosio::**minutes** (int64_t *m*)

inline *microseconds* eosio::**hours** (int64_t *h*)

inline *microseconds* eosio::**days** (int64_t *d*)

class eosio::**time_point**

High resolution time point in microseconds.

time_point (*microseconds elapsed* = *microseconds*())

microseconds &**time_since_epoch**()

uint32_t **sec_since_epoch**()

microseconds **elapsed**

The value used in serialization

class eosio::**time_point_sec**

A lower resolution time_point accurate only to seconds from 1970.

time_point_sec()

explicit time_point_sec (uint32_t *seconds*)

time_point_sec (**const** *time_point* &*t*)

time_point_sec **maximum**()

Maximum time_point_sec(0xffffffff)

time_point_sec **min**()

Minimum time_point_sec(0)

uint32_t **sec_since_epoch**()

Returns *utc_seconds*

uint32_t **utc_seconds**

The value used in serialization

class eosio::**block_timestamp**

1.11 Indices and tables

- genindex
- modindex
- search

INDEX

E

T