DataTokenAlpha Documentation

Zhuoqun Liu

Mar 20, 2018

CONTENTS

This is the documentation of DataTokenAlpha.

CHAPTER

ONE

INTRODUCTION

This project aims to develop a scheme for peer-to-peer (P2P) cellular data sharing and to construct a prototype Ethereum smart contract with respect to the scheme using Solidity which is a blockchain oriented programming language. Smart contract DataToken-Alpha (latest version 0.3.2) as the scheme implementation has been under development. Hence, DataToken, a token of Ether (Intrinsic Cryptocurrency of Ethereum) is created for data service transfer recording.The uniqueness of such implementation is that all the information held by the contract is protected by Ethereum network based on blockchain technology from computational hacking.

The implementation is started from analyzing behaviors that a ledger system is expected to have. By design, the contract is firstly able to offer and manage token like a bank in real world and secondly, it is capable to work as a cellular data service trade centre where users of such contract are able to transfer Internet access in terms of cellular data usage and token. Solidity code thud is used to implement necessary features with respect to the behaviors mentioned above. Finally, a prototype of DataToken-Alpha for autonomous data trading is ready for further polish in the next semester when this project will focus on upgrading DataToken-Alpha to be a practically deployable smart contract as close as possible.

CHAPTER TWO

TERMS

In the context of DataTokenAlpha contract, each Ethereum address has three possible roles:

- Receiver (Default role)
- Provider (A receiver can switch user role as provider)
- Paired (A receiver who is using AP service of a provider)

Note: Both "receiver" and "paired" are referred to as "receiver" in the following documentation. User *identification* value will distinguish paired receiver from unpaired receiver.

CHAPTER

THREE

CODING STYLE

Camel case names for variable and function names. Internal functions all start with an underscore _. Input parameter of functions starts with an underscore _.

CHAPTER

FOUR

CONTRACT VARIABLES

Note: Contracts variables are declared at the beginning of solidity contract body. In this section, variable types and availability are indicated by source code definition below each mentioned variables.

4.1 owner

```
address owner;
```

The message sender address of the transaction that has initiated this DataTokenAlpha contract is stored in this variable.

All initial supply tokens are assigned to this address. That is, all usable tokens are sold buy owner address.

4.2 tokenName

```
string public tokenName = "dataToken";
```

The name of token defined by this contract is dataToken.

4.3 decimals

```
uint public decimals = 9;
```

This token will support 9 digits after decimal point. The number of 9 actually defines the smallest operable unit of each token.

4.4 initialSupply

```
uint256 initialSupply = 666;
```

There will be 666 usable tokens in this contract.

4.5 totalSupply

```
uint256 totalSupply = initialSupply * 10 ** decimals;
```

Number of tokens in terms of the smallest unit is 666,000,000,000.

Note: The smallest unit is DataToken. There are 666,000,000,000 DataTokens supplied by this contract.

4.6 APID_counter

```
uint256 APID_counter = 1;
```

This number will be assigned to a newly registered provider.

If a successful call of *surProvider* is initiated by a user without *APID* value, APID_counter will be assigned to the new provider's *APID* mapping, and the value of APID_counter will be updated by +1 as *surProvider* implemented.

4.7 role

```
enum role {ISRECEIVER, ISPROVIDER, PAIRED}
```

This variable defines three possible roles of contract users.

Numerically, identification has values:

- role.ISRECEIVER = 0
- role.ISPROVIDER = 1
- role.PAIRED = 2

Note: The following variables are of mapping type.

Click to find mapping in solidity documentation

4.8 identification

```
mapping (address => role) public identification;
```

This mapping takes Ethereum address as key and role (enum type) as the mapped value.

By default, any unassigned value is recognized as 0, therefore, Ethereum addresses automatically have *role.ISRECEIVER* (numerical value is 0) as mapping values of *identification*.

When a receiver address calls function *surProvider*, *identification* mapping value of this address will be changed to *role.ISPROVIDER* (numerical value is 1).

When a receiver address has called function *link* successfully, mapping value of the address will be designated as *role.PAIRED* (numerical value is 2).

4.9 APID

```
mapping (address => uint256) public APID;
```

This mapping shows numerical ID of a provider address.

When a provider is deploying Wi-Fi AP, frontend client could query value of this mapping with the Ethereum address of the provider. Then the unique numerical ID can be shown in SSID.

There are two reasons for this mapping:

- SSID has String length limit. A full length Ethereum address exceeds such limitation, however, a truncated address is not easy to resolve.
- Ethereum address behind a wireless AP could be protected by this APID. Currently, *providerBehind* is publicly declared, but it's high availability is not necessary.

4.10 balance

```
mapping (address => uint256) public balance;
```

Mapping balance uses Ethereum address as key and number of tokens as value.

Token balance of each contract user can be viewed by calling this mapping.

Only one internal function *_transfer* can manipulate values of this mapping without restriction.

4.11 providerBehind

```
mapping (uint256 => address) public providerBehind;
```

This mapping is a conversed version of *APID* mapping.

APID of an account is a key of this mapping. The value corresponds to the key (APID grabbed from SSID) is the Ethereum address of the AP host.

4.12 numberOfUsers

```
mapping (address => uint) public numberOfUsers;
```

For each contract user of provider role, this mapping is important.

- When a receiver is linked to a provider by *link* function, mapping value of the provider should be added by 1.
- When the receiver has successfully called function *payAndLeave*, the value of numberOfUsers should be decreased by 1.
- Only when this mapping value is 0 which is the default value, can a provider call function *surReceiver* to switch user role back to *role.ISRECEIVER*.

4.13 providerOf

```
mapping (address => address) public providerOf;
```

For each user of *role.PAIRED* who was of *role.ISRECEIVER* before a successful call of *link* function, this mapping will be assigned by the Ethereum address of the linked provider.

Only users being served has nonzero providerOf mapping. Value of this mapping will be reset to 0 after a successful call of *payAndLeave*.

4.14 priceOf

```
mapping (address => uint256) public priceOf;
```

When a receiver intends to switch user role to be a provider, function *surProvider* will request a input that specifies pricing of this AP service to deploy in DataToken/MB.

4.15 usageOf

```
mapping (address => mapping (address => uint256)) public usageOf;
```

This is a mapping designed to verify data usage information to prevent cheating on both sides when issuing payment.

A function *_tolerance* is defined to check whether data usage record can reach a consensus.

If records from both provider and receiver agree with each other, the receiver will pay for the amount of data usage specified by the provider.

Warning: What will happen if a consensus is not reached has not been defined yet!

4.16 passwd

```
mapping (address => string) internal passwd;
```

This mapping is where provider can store their designated key to generate dynamic PIN for wireless AP authentication.

User will be require to input a password when function *surProvider* is called.

CHAPTER FIVE

EVENT

Note: Event is used as log when important information of the contract is changed, for example, user balance changed as a result of transfer.

5.1 Transfer

```
event Transfer(address _from, address _to, uint256 value);
```

Adding this event to the end of a function that issues token transfers will trigger a return message about the transfer.

5.2 sur

```
event Sur(address _user, role _newrole, bool success);
```

When a user switch user role, this event will return a message indicating the original user role, the intended user role and whether the operation has succeeded.

CHAPTER

SIX

INTERNAL FUNCTIONS

Note: Internal functions are invisible to Web3 Javascript API and external calls from other contracts.

Warning: Internal functions are defined with great power that can easily change data on Ethereum like DataToken balance of contract users.

6.1 _transfer

```
function _transfer(address _from, address _to, uint256 _value)
```

Warning: This function has great power that it can manipulate balance between addresses without any restriction.

A call of this function is able to transfer _value amount of token from Ethereum address _from to address _to. The function has the highest authority in a transfer operation.

It is defined as internal for safety concern that API (web3 implemented by ethereum core team) cannot call this function directly.

Tip: This function is the core of functions that are able to cause change of DataToken balance.

6.2 _sur

```
function _sur(address _user, role _oldrole, role _newrole)
```

"sur" represents switch user role. This function will not check current user role but will simply alter *identification* mapping of input *_user* address with input *_newrole*. Public functions that can switch user roles all depend on this internal function.

Tip: This function is the core of function *surReceiver* and function *surProvider*.

6.3 _affordableData

```
function _affordableData (address _wallet, uint256 _price)
```

This function is used to find data usage limitation for a receiver.

DataToken balance of this receiver and data service pricing of the linked provider are considered.

The output value can be used for data usage countdown.

Warning: Current version of contract is not ready for the countdown feature.

Tip: function *link* depends on this function because it requires receiver must have balance to pay for no less than 1 MB to call function *link*.

6.4 _tolerance

```
function _tolerance (uint256 _range, uint256 _usageLimit)
```

This function is used from receiver's perspective. When a receiver is about to quit AP service from the provider, this function checks mapping *usageOf* values of both the receiver and the provider to make sure they agree with each other within a tolerance defined as consensus of this contract.

Tip: function *payAndLeave* requires _tolerance to be true.

6.5 _cashier

```
function _cashier (address _payer, uint256 _volume)
```

This function is responsible to collecting payment from a receiver when the receiver calls function *payAndLeave*.

Tip: This function is an important component in function *payAndLeave*.

CHAPTER SEVEN

PUBLIC FUNCTIONS

Public functions can be invoked in other contracts and is callable by Web3 Javascript API.

7.1 DataTokenAlpha

```
function DataTokenAlpha() public {
    owner = msg.sender;
    balance[owner] = totalSupply;
}
```

Tip: This is the constructor function of solidity contract DataTokenAlpha.

7.2 transfer

```
function transfer(address _to, uint256 _value)
```

7.3 buyToken

```
function buyToken()
payable
public
returns(bool success)
{
    _transfer(owner, msg.sender, msg.value / 10 ** 9);
    return true;
}
```

This function is payable. That means, any contract user can call this function with a specified Ethereum value in terms of wei. The function will then transfer equivalent amount of DataToken from *owner* address to the buyer's address according to the exchange rule that 1 DataToken = 1,000,000,000 wei = 1 Gwei.

Warning: Thi function will only transfer Ether from buyer to contract rather than from buyer to *owner* to buy DataToken. There should be a way for the *owner* to deposit Ether from the contract.

7.4 surProvider

```
function surProvider (uint256 _price, string _passwd)
```

This function can only be called by contract users with *identification* role.ISRECEIVER. On success, *role.ISPROVIDER* will be assigned to identification mapping of the message sender.

7.5 surReceiver

```
function surReceiver (uint _numberOfUsers)
```

When a provider want to switch back to be a receiver, this function will be there for help. The provider will be required to have no linked receiver who is using wireless AP under the name of this provider. On success, *role.ISRECEIVER* will be assigned to *identification* mapping of the message sender.

7.6 link

```
function link (address _provider)
```

Only receivers with *role.ISRECEIVER* can call this function. This function will pair the message sender with the designated provider.

..warning:

```
This function actually relies on a address resolver since the frontend client should␣
↪only feed the function APID. And such resolver is not implemented.
```

7.7 usageRecord

```
function usageRecord (address _theOtherSide, uint256 _usage)
```

This function assigns value of *usageOf* in terms of MB.

Warning: There should be some timing and data refreshing features to make the mapped data up to date, however, this feature is not implemented within this version of contract.

7.8 payAndLeave

```
function payAndLeave (uint256 _range, uint256 _usageLimit)
```

When a receiver wish to leave it's wireless AP, it can call this function to issue a payment and disconnect from the provider.