Bigbom Digital Contracts Documentation

Release 9/2018

Chris Nguyen

Nov 19, 2018

Overview:

CHAPTER 1

Bigbom Digital Contract Platform Technical Paper

This technical paper describes how a contract is being established between parties, and how Bigbom Digital Contract Platform will implement it's idea to implement a decentralized digital advertising ecosystem via smart contracts.

1.1 A brief history on contract signing

Paper Contract: The oldest, most popular method for establishing business relationship. Usually it requires two people, representing their organization or themselves, in order to sign the contract. Some contracts requires a 3rd-party as the validator. Once signed, both parties is obliged to follow their duties, written in the contract. Whether a dispute happens, it will be resolve using the law of country that is designated in the contract.

One of the biggest problem with a paper contract is geography distance. Imagine Bob, which lives in New York and need to do business with a partner in South Africa, he has to travel approximately 15 hours in order to get to Johannesburg, took another day to meet the partner and sign to the contract, then take another same long flight back to New York. What happens if his partner suddenly changes his mind, refusing to sign into the contract? What happens if he's gone rouge, and ran off AFTER took your money? Is Bob gonna hire a lawyer to bring him to the court, and then paying all the expenses until he able to get the money back ?

With the development of internet communication, paper contract is less and less importance for a business, since it's show its weakness in cost and execution. That's why Digital Contract was invented

Electronic Contract aka Docusign and friends: With the development of internet applications and cryptographic algorithm, e-signature solutions was developed to give users the capability of signing contract without having physical presence. One of the leading solution of e-signature is Docusign, which provides a Software-as-a-Service signing solution. Signing a contract now requires just a few steps:

1. Uploading a contract under digital format, eg PDF or word format
2. Send an invitation link to signers. Each signer requires to create an e-signature
3. Sign the document. The signed document then available for both parties, with the e-signature stamped on the document.

With e-signature, Bob now does not need to spend 30 hours in airplane, and he still can do business with his partner in Johannesburg. But the problem still persist. There is no guarantee that his partner will follow his obligation in the contract. Geographic problem solved, but what about execution problem ?

1.2 The introduction of Bigbom Digital Contract

Bigbom Digital Contract is a blockchain-based solution built with the aim of solving the contract signing and contract execution problem using a single platform. By combining the power of blockchain and e-signature, Bigbom Digital Contract will help people doing business together, with less worries about the authenticity of the contract, and the payment part of the contract. Let's take this scenario to see how Bigbom Digital Contract is going to work.

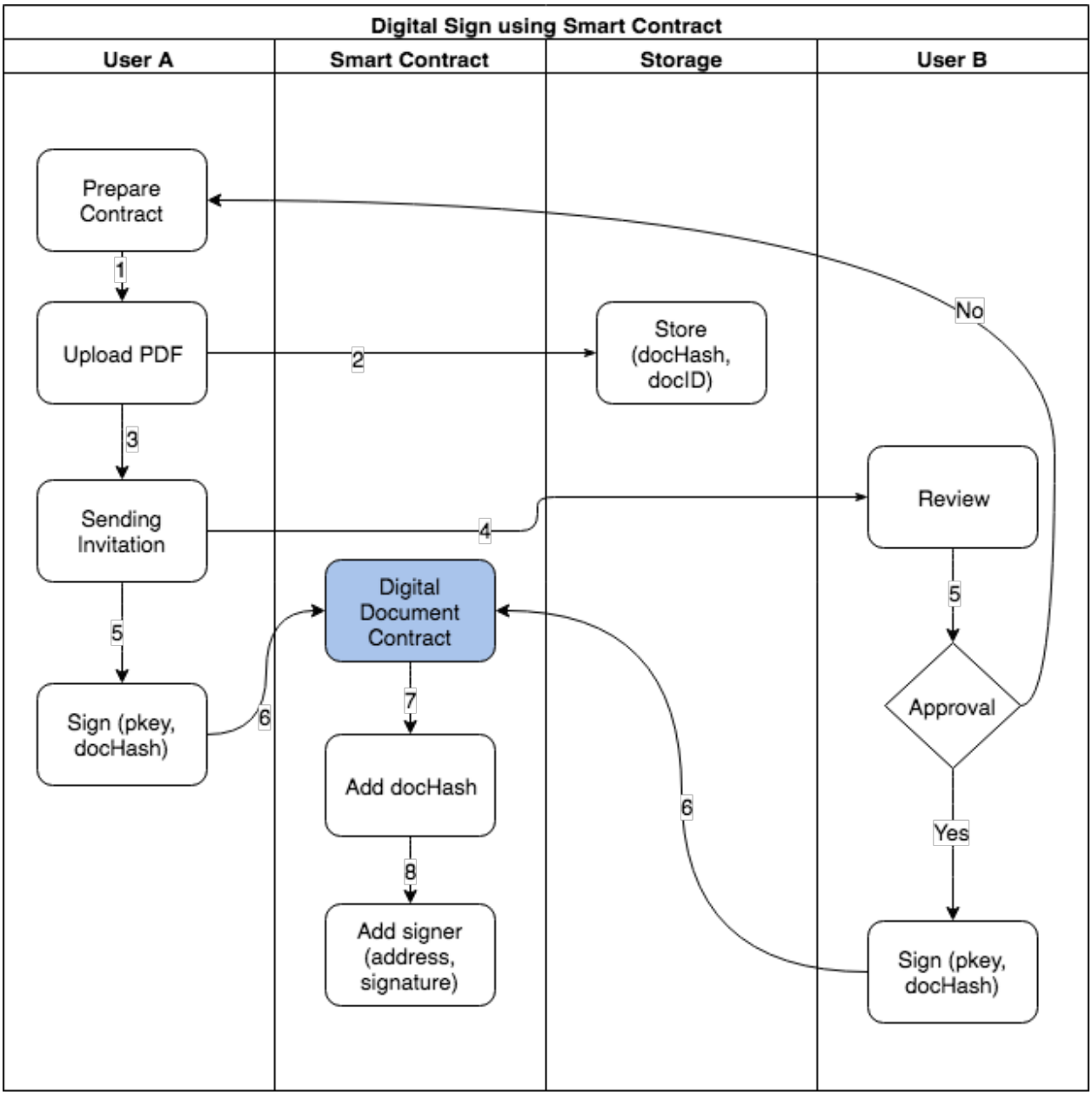
Mandy is an advertiser, and she wants to run an ads campaign on John's website, https://thefamousjohn.com. Through direct contact, John is offering Mandy a Cost-Per-Click campaign, with the total 100,000 clicks on a banner sitting on top of John's website. The cost per every click is agree at 0.05$, which means it will cost Mandy $5000 for the whole campaign, expected to run in 15 days.

Here is how Bigbom Digital Contract is going to take place.

1.2.1 E-Signing:

John will draft a contract, with all the terms included. After that he will start uploading the contract, using PDF format into Bigbom Digital Contract platform. John then send an invitation to Mandy to her email address, asking Mandy to review and sign the contract. After John and Mandy signed the contract, Bigbom Digital Contract platform will calculate the hash of the document and store it into blockchain using a smart contract. By using this, signed data is permanent, and both John and Mandy can re-check the authenticity of the contract anytime.

Below is the workflow for signing a document by using smart contract:



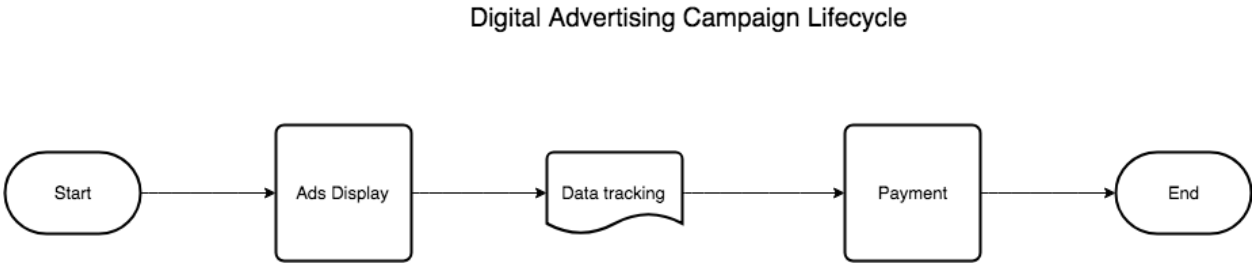
- docHash is a sha3 hash from uploaded pdf. Notes that the hash is not on the file, but on the contents itself. An example code in python

```
>>> Web3.sha3(0x747874)
>>> Web3.sha3(b'\x74\x78\x74')
>>> Web3.sha3(hexstr='0x747874')
>>> Web3.sha3(hexstr='747874')
>>> Web3.sha3(text='txt')
HexBytes('0xd7278090a36507640ea6b7a0034b69b0d240766fa3f98e3722be93c613b29d2e')
```

- Now with the docHash, a person with a Ethereum private key could easily sign it. The result will be a signature of the docHash. Only the owner of signed address is able to verify signature.
- An Ethereum smart contract is responsible for store and query the list of signer. By validating the signer list, we can know if a document has been signed by other parties or not.

1.2.2 Digital Contract Verification & Payment:

For contract execution, there are two key factors: Verifies that the contract has been executed as agreed, and proccessing the payment. For tracking the execution of the contract, Bigbom Digital Contract allows to update the contract progress via different methods, depends on the contract type. The general lifecycle is being describing in below image:



A vital part for both Advertisers and Ads Platform/Publisher is they have to able to know how much their budget has been spent, and how much the is the progress of the campaign. By using smart contract, both parties will be able to traceback to their historical data through the contract, since all data is immutable and transparent for both sides.

Digital Advertising Campaign type and metrics:

Cost-Per-Click Ads (CPC): A Cost-Per-Click Ad represents the cost to the advertiser everytime someone click to their ads. Most of online ads platforms requires a target cost-per-click for new campaign to run.

Cost per Click formula:

Cost-Per-Mile (CPM): The "cost per thousand advertising impressions" metric (CPM) is calculated by dividing the cost of an advertising placement by the number of impressions (expressed in thousands) that it generates. CPM is useful for comparing the relative efficiency of various advertising opportunities or media and in evaluating the overall costs of advertising campaigns (definition source)

Cost per Mile formula:

Cost-Per-Install (CPI): Cost Per Install campaigns are specific to mobile applications. In a Cost Per Install campaign, publishers place digital ads across a range of media in an effort to drive installation of the advertised application. The brand is charged a fixed or bid rate only when the application is installed. (definition source)

Cost per Install formula:

Contract-As-Code:

When an advertiser wants to buy a slot for their campaign, they will have to consider about their ***target*** and ***budget*** , meaning that they will select which network that is able to provide the maxium target with the fixed budget cost. In some other cases, some advertisers just want to reach their ***target***, and they're willing to spend the exact amout of budget that is able to reach their target. Let's try to imagine if a digital ad contract is a computer software function with a set of parameters, that function will likely has these parameters:

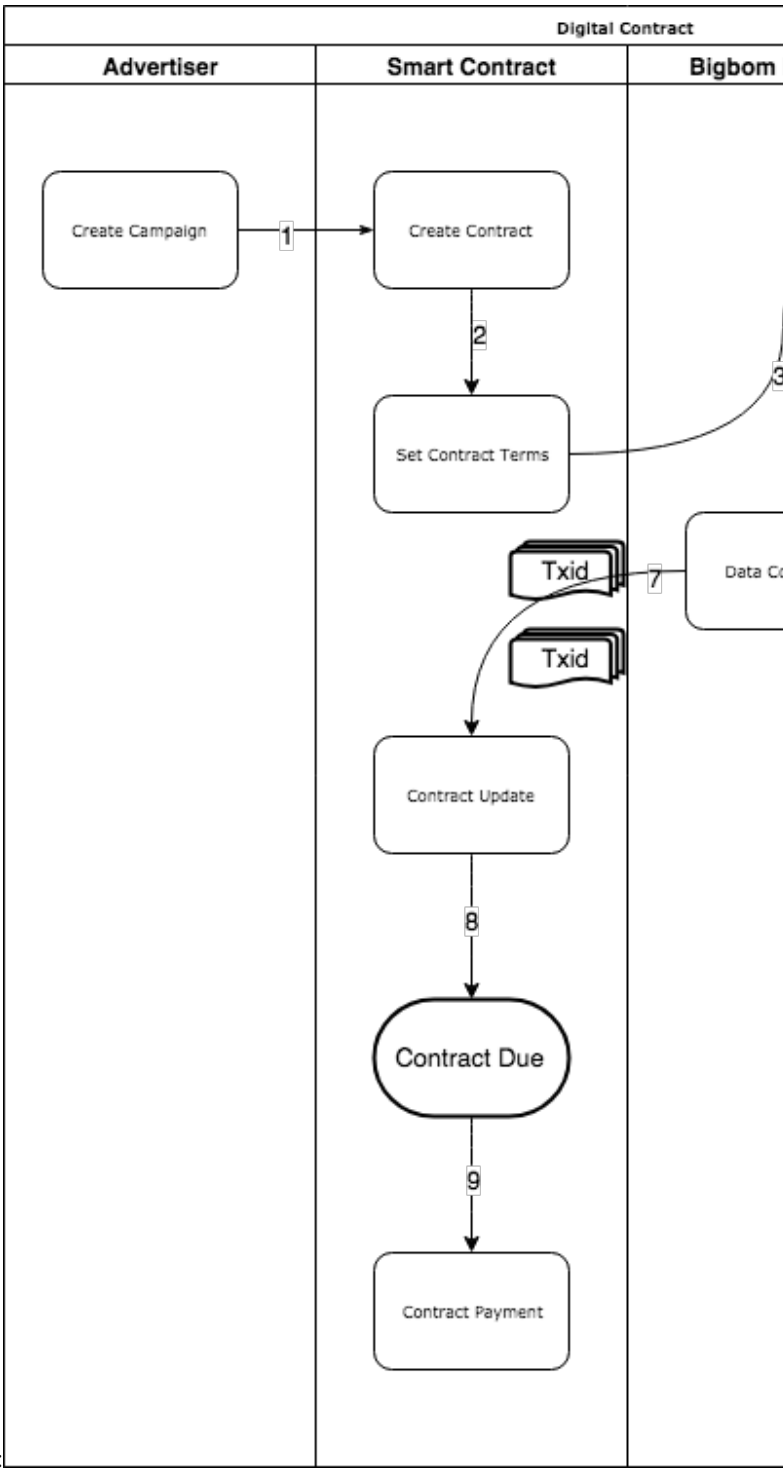
1. Signed copy of the contract terms (as legal material)
2. Contract start date
3. Contract end date
4. Agreed target (amount of clicks, impressions or installs)
5. Budget
6. Minimum guaranteed target (for contract violation term)
7. Current milestone (current amount of clicks, impressions or installs)
8. Current spending

With a contract-as-code design, it's very easy to both advertisers and ads platforms/publisher to track and viewing the progress. In practice, many ads platform and publisher is already implementing their own system to tracking and updating their progress to their customers. However not all system is bullet-proof with cybersecurity crime, and the cost of keeping data availability is not cheap at all. For a long time, most of enterprises is depending into expensive systems/storages in order to achieving this.

> Whereas most technologies tend to automate workers on the periphery doing menial tasks, blockchains automate away the center. Instead of putting the taxi driver out of a job, blockchain puts Uber out of a job and lets the taxi drivers work with the customer directly.
>
> Vitalik Butterin

By using smart contract, all of these parameters will be stored inside the smart contract. Since blockchain is immutable by itself, it's very easy for both parties to query all the historical data (through the transaction id) and validates with the actual results. Bigbom Digital Contract provides a set of contract templates that suits to each scenario, whether you want to run your campaign with a specific *target*, or you want to join to a real-time bidding system with a specific amount of *budget*.

Once these terms has been put into the smart contract. Bigbom Digital Contract platform will start to monitor the campaign, and consequentaly update the campaign into the smart contract, until it ends. At this point, both Mandy and John is aware about how well the campaign was, and what should be the actual cost.



Following workflow is describing how contract-as-code works:

Payment: Payment is a very complex topic. With a Digital Advertising Campaign, the actual bill is depends on various factor, like the actual amount of clicks or installations.

We're currently developing our smart contract in order to support following payment method:

Due Date Payment: The final payment amount will be determine at the contract end date. Based on the current result, Our smart contract will determine the exact amount of payment for the seller side.

Pay as you go: A prefer payment method for most online platform is charging their customer based on the progress.

Currently our smart contract is supporting daily payment. The amount of payment is determined by using the progress of previous day, for calculating how much the seller side will get.

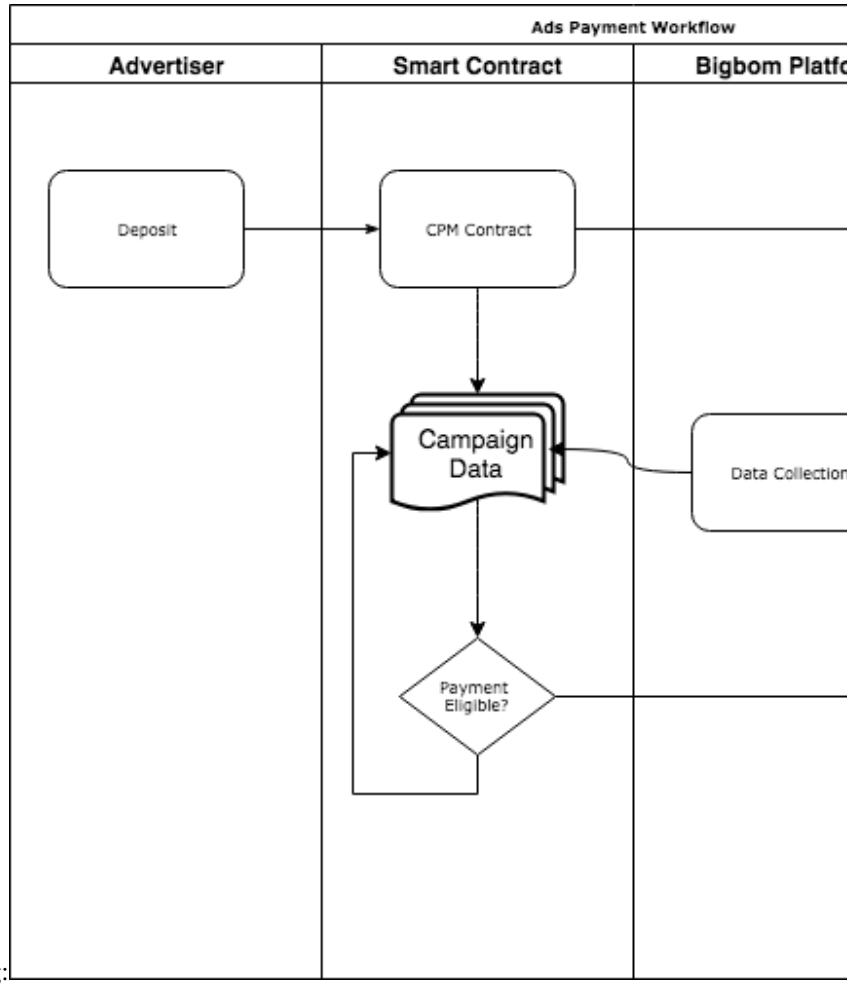
Generally speaking, the contract will end up in these scenarios:

a. John get paid based on what he delivered, for example $3500 for 70,000 clicks after 15 daysb. John get paid full amount of $5000, if he able to deliver 100,000 clicks or more after 15 daysc. John does not get paid at all, since he's only able to deliver 9,000 clicks, and in the contract they both agreed that if the target dropped below 10%, Mandy won't have to pay anything.

Bigbom Digital Contract is able to adapt with all these scenarios. Before the contract goes into effective, it is required that Mandy needs to make a security deposit into the contract. The amount of security deposit is negotiable between Mandy and John. Bigbom Digital Contract will evaluate the amount of deposit and calculate the worth of clicks or installations that is equal to the amount of desposit. Since the campaign stats is constanly being monitored by Bigbom Digital Contract Platform, Mandy and John is able to see how much money has spent for the campaign. Before the deposit is being depleted, Bigbom Digital Contract Platform will notify Mandy for making another deposit, in order to keep the campaign running. If Mandy refuses to make the deposit, John will get a notification, suggesting him to suspend the campaign.

Even if Mandy keeps the commitment and send the security deposit, there is a possibility that John is not able to fulfill the contract terms, for example he cannot deliver more than 10,000 clicks after 15 days. In this case, since all the security deposit is being kept by the Digital Contract, John will receive any money, and at the end of the campaign, Mandy will get a refund, if the refund condition is met.

A demonstration for this payment process is as following:



Fiat/Token conversion: One of the biggest obstacle for people when using blockchain products is volatility in price.

Imagine a token with a price $0.02 today become $0.05 tomorrow, and then drops back to $0.01 the day after. With the majority of people still using fiat currencies for daily trading, this volatility is unacceptable and prevent them to adopting blockchain products. With the aim to create an entrance for fiat users, Bigbom Digital Contract Platform is developing a method that calibrates the amount of actual BBO Token should be paid before the actual payment happens, so people will be kept away from the volatility in price. Here's how it works

Scenario 1: Mandy is a BBO hodler, and John acccept BBO for his payment. This is a very straight-forward case, Mandy will make the deposit by BBO, and John will get paid by BBO as well.

Scenario 2: Mandy and John is not BBO holder, and they want to settle the contract in fiat, for example USD Dollar. Bigbom Digital Contract Platform will ask Mandy for making the deposit in fiat, by sending money via a payment gateway. After receiving the deposit, Bigbom Digital Contract Platform will calculate it to the actual BBO Token amount based on the market price, and took an amount of BBO equivalent to the original amount, multiplying with a factor of 2 from the Bigbom Reserve, then sending it to the Digital Contract as the deposit. The reason behind multiplying to a factor to ensuring that Bigbom Digital Contract platform is able to endure the price changes up to two times on market price each time payment term is being triggered.

Once the contract finished, Bigbom Digital Contract Platform will re-calculate the actual amount of BBO needed to pay, based on John's performance and current market price. At the end of the day, John will receive the amount of BBO with similar value into USD Dollar that he supposed to get from Mandy. By integrating with other exchange platform, John will have the capability to swap BBO into Ethereum/Bitcoin, or even Fiat currencies after he received it.

At current phase, we will support USD and SGD as accepted fiat currencies, and will integrate with Kyber Network for swapping BBO to ETH. More currencies and swapping method will come in the future.

CHAPTER 2

Freelancer Smart Contract Architecture

2.1 Introduction

There are three main components in Bigbom Freelancer App: `BBFreelancerBid.sol`, `BBFreelancerJob.sol`, `BBFreelancerPayment.sol`

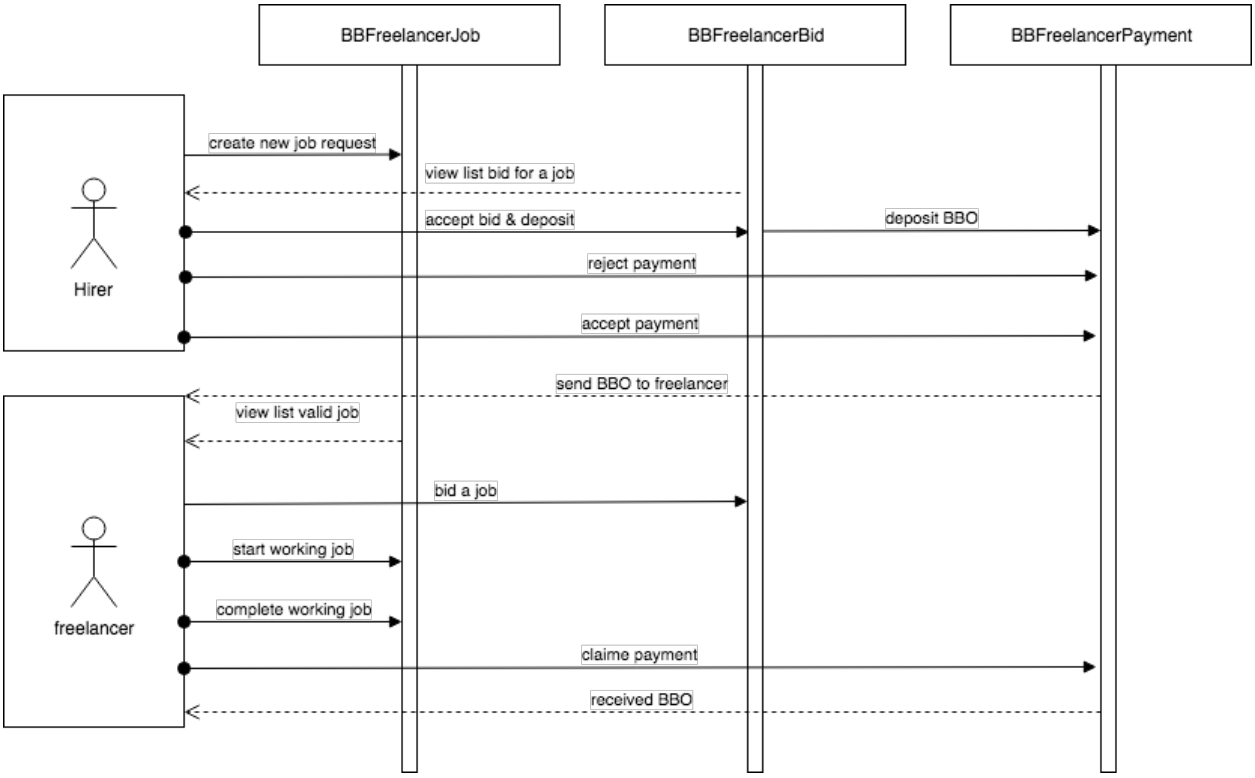
`BBFreelancerJob.sol`: manage job information

`BBFreelancerBid.sol`: manage biding

`BBFreelancerPayment.sol`: manage payment

2.2 High-level Overview

BBOFreelancer Dapp allows the hirer to post the job, they will automatically begin to receive bids from the freelancers. They can choose the best freelancer & success work.



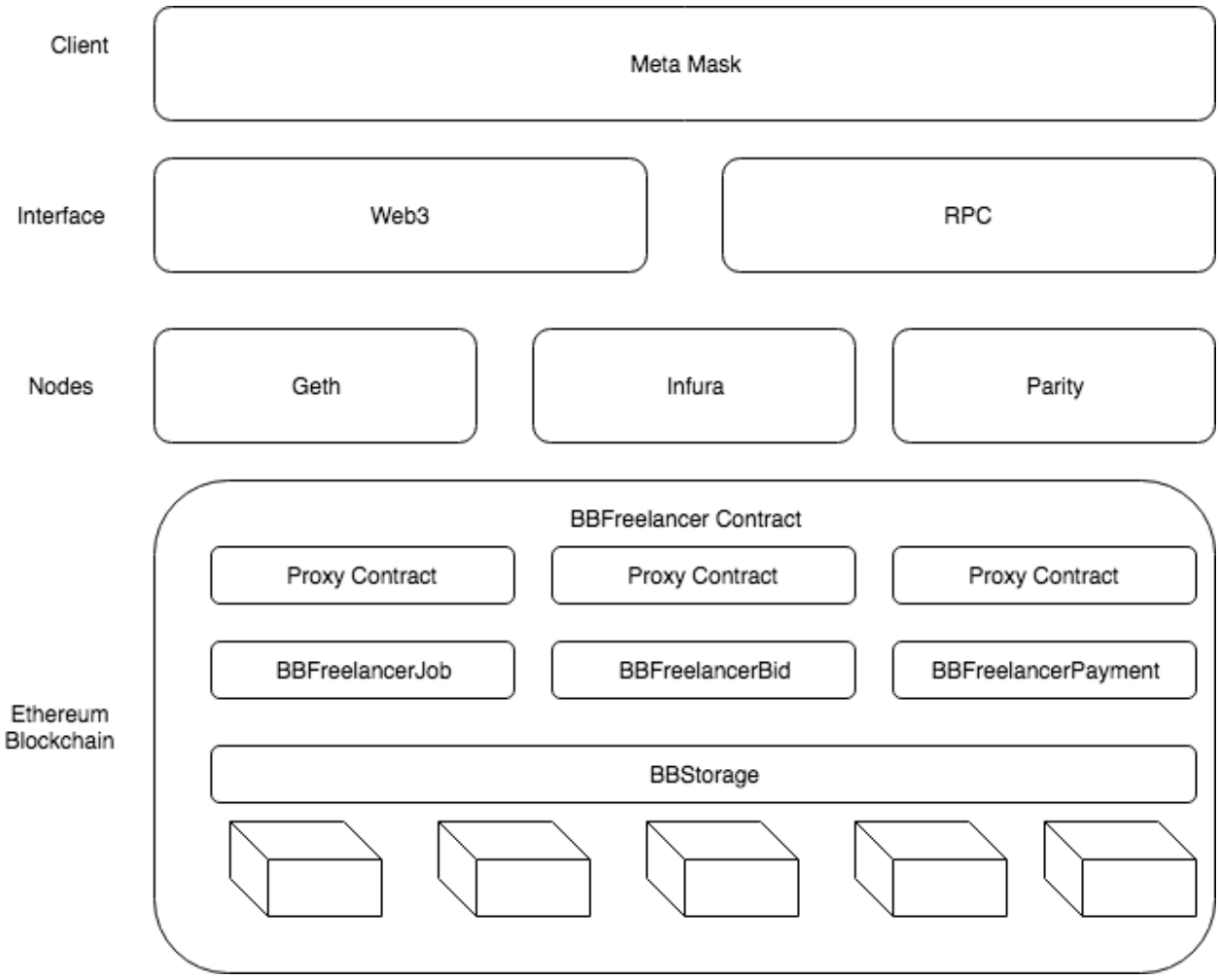
2.3 Detailed Hirer Flow

- Post a job
- Choose the perfect freelancer
- Pay & success

2.4 Detailed Freelancer Flow

- Find a job
- Make your best bid
- Get awarded and earn

2.5 Stack



2.6 Permissions

BBFreelancer always has proxy contract over each contract to make the contract upgradeability. They are has `Admin permision` is allowed to write to `BBStorage` contract

CHAPTER 3

How to work with Bigbom Freelancer DApp

3.1 Ropsten Contract Information

- BBO: `0x1d893910d30edc1281d97aecfe10aefeabe0c41b`
- proxyAddressJob: `0x62aa93f9dffec25daf9d2955d468194e996e8c87`
- proxyAddressBid: `0x0ff11890ef301dfd0fb37e423930b391836c69c9`
- proxyAddressPayment: `0x7b7e6f2b02a48bd24b5b1554fafff5f70547ab0a`

3.2 Example

- Job Status 0 : init 1 : start 2 : finish 4 : reject 5 : claim Payment 9 : accept Payment
- Create new job

```
let job = await BBFreelancerJob.at(proxyAddressJob);
var userA = '0x12312321';
var expiredTime = parseInt(Date.now()/1000) + 7 * 24 * 3600; // expired after 7 days
var totalTime = 3 * 24 * 3600;// requirement job done in 3 days
// createJob: jobHash, expiredTime, totalTime,budget, category
var jobLog  = await job.createJob(jobHash, expiredTime, totalTime, 500e18, 'banner',
↪{from:userA});
// check event logs
```

- cancel job

```
    var jobLog  = await job.cancelJob(jobHash, {from:userA});
```

- create bid

```
    var userB = '0x98988997897';
    let bid = await BBFreelancerBid.at(proxyAddressBid);
    var timeDone = 2 * 24 * 3600; //Freelancer suggest time done affer 2 days
    var jobLog  = await bid.createBid(jobHash, 400e18, timeDone,{from:userB});
```

- cancel bid

```
    var jobLog  = await bid.cancelBid(jobHash, {from:userB});
```

- acceept bid

```
    var userA = '0x12312321';
    let bid = await BBFreelancerBid.at(proxyAddressBid);
    let bbo = await BBO.at(BBOAddress);// abi BBO contract
    // approve bid's BBO amount to contract proxyAddressBid
    await bbo.approve(proxyAddressBid, 400e18, {from:userA});
    // call accept function
    var jobLog  = await bid.acceptBid(jobHash, accounts[1], {from:userA});
```

- start working job

```
    var jobLog  = await job.startJob(jobHash, {from:userB});
```

- finish job

```
    var jobLog  = await job.finishJob(jobHash, {from:userB});
```

- reject payment

```
    let payment = await BBFreelancerPayment.at(proxyAddressPayment);
    var userA = '';
    var jobLog  = await payment.rejectPayment(jobHash, {from:userA});
```

- claime payment

```
    let payment = await BBFreelancerPayment.at(proxyAddressPayment);
    var userB = '';
    var jobLog  = await payment.claimePayment(jobHash, {from:userB});
```

- acceept payment

```
    var jobLog  = await payment.acceptPayment(jobHash, {from:userA});
```

- get job by Hash

```
    let job = await BBFreelancerJob.at(proxyAddressJob);
    var jobLog  = await job.getJob(jobHash);
    // return [owner, expired, budget, cancel, status, freelancer]
```

- view list job

```
//  event JobCreated(bytes jobHash, address indexed owner, uint created, string␣
↪category);

BBFreelancerJob.at(proxyAddressJob).getPastEvents('JobCreated', {
    filter: {owner: '0x123', category: ['banner','it']},  // filter by owner, category
    fromBlock: 0, // should use recent number
```

(continues on next page)

(continued from previous page)

```
    toBlock: 'latest'
}, function(error, events){
    //TODO
    });
```

3.3 Event lists:

- event JobCreated(bytes jobHash, address indexed owner, uint expired, bytes32 indexed category, uint256 budget);
- event JobCanceled(bytes jobHash);
- event JobStarted(bytes jobHash);
- event JobFinished(bytes jobHash);
- event BidCreated(bytes jobHash, address indexed owner, uint256 bid, uint created);
- event BidCanceled(bytes jobHash, address indexed owner);
- event BidAccepted(bytes jobHash, address indexed freelancer);
- event PaymentClaimed(bytes jobHash, address indexed sender);
- event PaymentAccepted(bytes jobHash, address indexed sender);
- event PaymentRejected(bytes jobHash, address indexed sender);

CHAPTER 4

Ethereum Mainnet

CHAPTER 5

Ropsten Testnet

```
var proxyAddressJob = '0xb1e878028d0e3e47c803cbb9d1684d9d3d72a1b1'
var proxyAddressBid = '0x7b388ecfec2f5f706aa34b540a39e8c434cfc8b4'
var proxyAddressPayment = '0x253f112b946a72a008343d5bccd14e04288ca45c'
var storageAddress = '0x99a2c9bc3793cc72a7a9b352e97deece4f4961c7'

var proxyFactAddress = '0xac141d2fa2740bd57e5c035a811b6fb6cceb4b71'
var paramsProxy = '0xc0647055b50dce8751908bfbd7f1d219ed592d6f'
var disputeProxy = '0x2b44a5589e8b3cd106a7542d4af9c5eb0016ef6e'
var votingProxy = '0xc7252214d78b15f37b94ae73027419a9f275c36f'
var bbo = '0x1d893910d30edc1281d97aecfe10aefeabe0c41b'
var votingHelperProxy = '0x771911025b4eafb6395042b7dca728b275e5d8c0'
```

CHAPTER 6

Rinkeby Testnet

```
proxyAddressJob 0x71356605e4f79fd07b01cc187bdcbc1f4025db1f
proxyAddressBid 0xf01cc898b9245930a345bec82423b87f602cb8e4
proxyAddressPayment 0x22ce61d3c44e5a005a9b9f4485cfbc660c1c2ef3
BBOAddress 0x2ddc511802a37039c42c6bdb36028b2f8992b0fe
BBStorage: 0x7f4f85ed6fb35be5ab03272f95b73dfe4b491243

var proxyFactAddress = '0x46820d60ca35cab8103a332804c8c889358ec66f';

BBOFaucet: 0xab4c08e651c709644a58a6ebd8e1be3afa6aa34c

paramsProxy 0xb1b1e7f9223bca9d66aa97b773935d4aec13165d
disputeProxy 0x278636913d5203a057adb7e0521b8df9431bdaa5
votingProxy 0x54a7cb877948518444e4c97c426cf47718ac94c3
```

CHAPTER 7

Tomochain

```
var proxyAddressJob = '0x57d83ffc5b5bacb5ed7dcbd86303c36c1e90080f'
var proxyAddressBid = '0x5c16dd2b3d444004b842a6fec13fab9d0d3bdc7f'
var proxyAddressPayment = '0x326486e00ed560538ec19887851fdfdf8dec40bb'
var storageAddress = '0xe67dc1a983d213b8f7302e21b883afff9fc588d7'
var proxyFactAddress = '0x45928f6ce178ce80874183038098b89f9fc59cfd'

var BBOAddress = '0x343aa154194f5d99cc28598b009ec23723b5a439'

paramsProxy = '0x315cf0a2c4e5dbee4cd8cf25c55d95b4badb364b'
disputeProxy = '0xdb5134f53d003d478a71973543f187304097b039'
votingProxy = '0x9899fcb82031f8cba60eb13d6e93e88365256612'

BBOFaucet: 0x1504d8a81492c60e47aeba16e89a7b04d23c6261
```

CHAPTER 8

DApps Integration Guide

CHAPTER 9

BBFreelancer

Contract **BBFreelancer** is *BBStandard*

imports: BBStandard.sol

Source: BBFreelancer.sol

BBFreelancer is modifiers contract used for `BBFreelancerBid`, `BBFreelancerJob`, `BBFreelancerPayment`

9.1 Modifiers

9.1.1 jobNotExist

Require job hash not exist in BBFreelancer System

---

modifier jobNotExist(bytes jobHash)

9.1.2 isFreelancerOfJob

Require the sender is the freelancer of this `Job ID`

---

modifier isFreelancerOfJob(bytes jobHash)

9.1.3 isNotOwnerJob

Require the sender is not the owner of this `Job ID`

---

modifier isNotOwnerJob(unit256 jobID)

9.1.4 isOwnerJob

Require the sender is the owner of this `Job ID`

---

modifier isOwnerJob(unit256 jobID)

9.1.5 isNotCanceled

Require this `Job ID` is not canceled yet

---

modifier isNotCanceled(jobID uint256)

9.1.6 jobNotStarted

Require this `Job ID` is not started yet

---

modifier jobNotStarted(bytes uint256)

CHAPTER 10

BBFreelancerJob

Contract **BBFreelancerJob** is *BBFreelancer*

imports: BBFreelancerPayment.sol, BBLib.sol, BBFreelancer.sol,BBRatingInterface.sol

Source: BBFreelancerJob.sol

BBFreelancerJob is the contract implements Job Posting actions for Freelancer app

- *Events*
  - *JobCreated*
  - *JobCanceled*
  - *JobStarted*
  - *JobFinished*
- *Functions*
  - *getJob*
  - *createJob*
  - *cancelJob*
  - *startJob*
  - *finishJob*
  - *getJobID*
  - *allowRating*

10.1 Events

10.1.1 JobCreated

Event for logging new Job creations.

---

event JobCreated(bytes jobHash, uint256 indexed jobID, address indexed owner, uint expired, bytes32 indexed category, uint256 budget, uint256 estimateTime);

10.1.2 JobCanceled

Event for logging canceled job.

---

event JobCanceled(bytes jobHash);

10.1.3 JobStarted

Event for logging started job.

---

event JobStarted(bytes jobHash);

10.1.4 JobFinished

Event for logging finished job.

---

event JobFinished(uint256 jobID);

10.2 Functions

10.2.1 getJobID

Get jobID by jobHash

---

function getJobID(bytes jobHash) public view returns(uint256)

Returns:

10.2.2 getJob

Get job detail by job hash.

---

function getJob(uint256 jobID) public view returns(address, uint256, uint256, bool, uint256, address)

Returns:

10.2.3 createJob

Post new job

---

function createJob(bytes jobHash, uint expired ,uint estimateTime, uint256 budget, bytes32 category) public jobNotExist(jobHash)

Modifiers: `jobNotExist`

10.2.4 cancelJob

Cancel a job by jobHash

---

function cancelJob(bytes jobHash) public isOwnerJob(jobHash)

Modifiers: `isOwnerJob`

10.2.5 startJob

Start working on a job by jobHash

---

function startJob(uint256 jobID) public isNotCanceled(jobID) jobNotStarted(jobID) isFreelancerOfJob(jobID)

Modifiers: `isNotCanceled`, `jobNotStarted`, `isFreelancerOfJob`

10.2.6 finishJob

Finish working on a job by jobHash

---

function finishJob(uint256 jobID) public isNotOwnerJob(jobID) isFreelancerOfJob(jobID)

Modifiers: `isNotOwnerJob`, `isFreelancerOfJob`

10.2.7 allowRating

Check rule Rating of user

---

function allowRating(address sender ,address rateTo, uint256 jobID) public view returns(bool)

Implement: `allowRating`

10.2.8 status

Job status mapping

CHAPTER 11

BBFreelancerBid

Contract **BBFreelancerBid** is *BBFreelancer*

imports: BBFreelancerPayment.sol, BBLib.sol, BBFreelancer.sol

Source: BBFreelancerBid.sol

BBFreelancerBid is the contract implements Bidding actions for Freelancer app

11.1 Events

11.1.1 BidCreated

Event for logging Bid creations.

---

event BidCreated(uint256 indexed jobID , address indexed owner, uint256 bid, uint256 bidTime)

11.1.2 BidCanceled

Event for logging the canceled Bid .

---

event BidCanceled(uint256 indexed jobID, address indexed owner);

11.1.3 BidAccepted

Event for logging Bid creations.

---

event BidAccepted(uint256 indexed jobID , address indexed freelancer);

11.2 Functions

11.2.1 setPaymentContract

Set the address of the FreelancerPayment contract. Only invoked by owner.

---

function setPaymentContract(address paymentAddress) onlyOwner public

11.2.2 createBid

Allow the freelancer to create new bid for job hash.

---

function createBid(uint256 jobID, uint256 bid, uint bidTime) public isNotOwnerJob(jobID) isNotCanceled(jobID) jobNotStarted(jobID)

Modifiers: `isNotOwnerJob`, `isNotCanceled`, `jobNotStarted`

11.2.3 cancelBid

Allow the freelancer to cancel the bid by job hash.

---

function cancelBid(uint256 jobID) public isNotOwnerJob(jobHash)

Modifiers: `isNotOwnerJob`

11.2.4 acceptBid

Allow the hirer to accept the bid for job hash.

---

function acceptBid(bytes jobHash, address freelancer) public

CHAPTER 12

BBFreelancerPayment

Contract **BBFreelancerPayment** is *BBFreelancer*

imports: BBFreelancer.sol, BBLib.sol

Source: BBFreelancerPayment.sol

BBFreelancerPayment is the contract control the payment for Freelancer app

12.1 Events

12.1.1 PaymentClaimed

Event for loging payment claimed.

---

event PaymentClaimed(uint256 jobID, address indexed sender);

12.1.2 PaymentClaimed

Event for loging payment accepted.

---

event PaymentAccepted(uint256 jobID, address indexed sender);

12.1.3 PaymentRejected

Event for loging payment rejected.

---

event PaymentRejected(bytes32 indexed indexJobHash, address indexed sender, uint reason, uint256 rejectedTimestamp, bytes jobHash);

12.1.4 DisputeFinalized

Event for loging payment claim.

---

event DisputeFinalized(uint256 jobID, address indexed winner);

12.2 Functions

12.2.1 acceptPayment

Hirer accept the payment when the freelancer done the job

---

function acceptPayment(uint256 jobID) public isOwnerJob(jobHash)

Modifiers: `isOwnerJob`

12.2.2 rejectPayment

Hirer reject the payment when the freelancer done the job

---

function rejectPayment(bytes jobHash, uint reason) public isOwnerJob(jobHash)

Modifiers: `isOwnerJob`

12.2.3 claimePayment

The freelancer can claim the payment if the hirer does not accept/reject after X duration.

---

function claimePayment(uint256 jobID) public isFreelancerOfJob(jobHash)

Modifiers: `isFreelancerOfJob`

12.2.4 checkPayment

The freelancer can check the payment of this job for ready claim

---

function checkPayment(uint256 jobID) public view returns(uint256, uint256)

Returns:

12.2.5 finalizeDispute

Finalize dispute job and send payment for the winer

---

function finalizeDispute(bytes jobHash) public returns(bool)

12.2.6 refundBBO

refund token to hirer if canceled

---

function refundBBO(bytes jobHash) public returns(bool) {

CHAPTER 13

BBStorage

Contract **BBStorage** is *Ownable*

imports: Ownable.sol

Source: BBStorage.sol

BBStorage is key-value type storage:

```
    mapping(bytes32 => uint256)    private uIntStorage;
    mapping(bytes32 => string)     private stringStorage;
    mapping(bytes32 => address)    private addressStorage;
    mapping(bytes32 => bytes)      private bytesStorage;
    mapping(bytes32 => bool)       private boolStorage;
    mapping(bytes32 => int256)     private intStorage;
```

13.1 Modifiers

13.1.1 onlyAdminStorage

Only allow access from the admin storage mapping, use for write/delete data

---

modifier onlyAdminStorage()

13.2 Events

13.2.1 AdminAdded

Event for logging admin additions or removals from the storage contract.

---

event AdminAdded(address indexed admin, bool add)

13.3 Functions

13.3.1 addAdmin

add/delete admin to allow write/delete storage object. Only owner can invoke.

---

function addAdmin(address admin, bool add) public onlyOwner

modifier: onlyOwner

13.3.2 getAddress

Get `address` value from storage mapping by key

```
function getAddress(bytes32 _key) external view returns (address)
```

Returns: `address` value

13.3.3 getUint

Get `uint256` value from storage mapping by key

```
function getUint(bytes32 _key) external view returns (uint256)
```

Returns: `uint256` value

13.3.4 getString

Get `string` value from storage mapping by key

```
function getString(bytes32 _key) external view returns (string)
```

Returns: `string` value

13.3.5 getBytes

Get `bytes` value from storage mapping by key

```
function getBytes(bytes32 _key) external view returns (bytes)
```

Returns: `bytes` value

13.3.6 getBool

Get `bool` value from storage mapping by key

```
function getBool(bytes32 _key) external view returns (bool)
```

Returns: `bool` value

13.3.7 getInt

Get `int` value from storage mapping by key

```
function getInt(bytes32 _key) external view returns (int)
```

Returns: `int` value

13.3.8 setAddress

Set `address` value to storage mapping by key

```
function setAddress(bytes32 _key, address _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.9 setUint

Set `uint256` value to storage mapping by key

```
function setUint(bytes32 _key, uint256 _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.10 setString

Set `string` value to storage mapping by key

```
function setString(bytes32 _key, string _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.11 setBytes

Set `bytes` value to storage mapping by key

```
function setBytes(bytes32 _key, bytes _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.12 setBool

Set `bool` value to storage mapping by key

```
function setBool(bytes32 _key, bool _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.13 setInt

Set `int` value to storage mapping by key

```
function setInt(bytes32 _key, int _value) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.14 deleteAddress

delete `address` value from storage mapping by key

```
function deleteAddress(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.15 deleteUint

delete `unit256` value from storage mapping by key

```
function deleteUint(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.16 deleteString

delete `string` value from storage mapping by key

```
function deleteString(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.17 deleteBytes

delete `bytes` value from storage mapping by key

```
function deleteBytes(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.18 deleteBool

delete `bool` value from storage mapping by key

```
function deleteBool(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

13.3.19 deleteInt

delete `int` value from storage mapping by key

```
function deleteInt(bytes32 _key) onlyAdminStorage external
```

Modifier: `onlyAdminStorage`

CHAPTER 14

BBDocumentSign

CHAPTER 15

BBLib

CHAPTER 16

BBStandard

Contract **BBStandard** is *Ownable*

imports: Ownable.sol, SafeMath.sol, BBStorage.sol, ERC20.sol

Source: BBStandard.sol

BBStandard is standard contract implements the key-value storage from `BBStorage`, and use `BBO ERC20` token for payment

16.1 Functions

16.1.1 setStorage

set Storage contract address. Only owner can invoke.

---

function setStorage(address storageAddress) onlyOwner public

modifier: onlyOwner

16.1.2 setBBO

set BBO token contract address. Only owner can invoke.

---

function setBBO(address BBOAddress) onlyOwner public

modifier: onlyOwner

16.1.3 withdrawTokens

withdraw any token in the contract. Only owner can invoke.

---

function withdrawTokens(ERC20 anyToken) public onlyOwner

modifier: onlyOwner

CHAPTER 17

BBParams

CHAPTER 18

BBDispute

Contract **BBDispute** is *BBStandard*

imports: BBStandard.sol, BBLib.sol, BBFreelancerPayment.sol

Source: BBDispute.sol

BBDispute is the contract implements Poll creation actions for creating dispute in Freelancer app

- *Events*
  - *PollStarted*
  - *PollAgainsted*
  - *PollFinalized*
  - *PollWhiteFlaged*
  - *PollExtended*
- *Functions*
  - *setPayment*
  - *isAgaintsPoll*
  - *startPoll*
  - *againstPoll*
  - *getPoll*
  - *finalizePoll*
  - *whiteflagPoll*
  - *extendPoll*

18.1 Events

18.1.1 PollStarted

Event for logging start new poll.

---

event PollStarted(uint256 jobID, address indexed creator);

18.1.2 PollAgainsted

Event for logging against the exist poll.

---

event PollAgainsted(uint256 jobID, address indexed creator);

18.1.3 PollFinalized

Event for logging against the exist poll.

---

event PollFinalized(uint256 jobID, uint256 jobOwnerVotes, uint256 freelancerVotes, bool isPass);

18.1.4 PollWhiteFlaged

Event for logging White-Flaged.

---

event PollWhiteFlaged(uint256 indexed jobID, address indexed creator);

18.1.5 PollExtended

Event for logging Extend a Voting duration.

---

event PollExtended(uinit56 indexed jobID);

18.2 Functions

18.2.1 setPayment

set Payment contract address. Only owner can invoke.

---

function setPayment(address p) onlyOwner public

modifier: onlyOwner

18.2.2 isAgaintsPoll

Check this Poll started for the job Hash has againts or not

---

function isAgaintsPoll(uint256 jobID) public constant returns(bool)

Return: (Bool)

18.2.3 startPoll

Create a Poll to start Dispute by provide the evident proofHash

---

function startPoll(uint256 jobID, bytes proofHash) public

18.2.4 againstPoll

Against a Poll to start Dispute by provide the evident proofHash.

---

function againstPoll(unit256 jobID, bytes againstProofHash) public

18.2.5 getPoll

Get Poll detail

---

function getPoll(uint256 jobID) public constant returns (uint256, uint256, bool)

Returns:

18.2.6 finalizePoll

Finalize a Poll

---

function finalizePoll(uint256 jobID) public

18.2.7 whiteflagPoll

White-flag a Poll

---

function whiteflagPoll(uint256 jobID) public

18.2.8 extendPoll

Extend a Poll

---

function extendPoll(uint256 jobID) public

CHAPTER 19

BBVoting

Contract **BBVoting** is *BBStandard*

imports: BBStandard.sol, BBLib.sol

Source: BBVoting.sol

BBVoting is the contract implements Partial-Lock Commit-Reveal Voting for allow voter can help to reslove the dispute in Freelancer app

- *calcReward*

19.1 Events

19.1.1 VotingRightsGranted

Event for logging the voter request the voting rights.

---

event VotingRightsGranted(address indexed voter, uint256 numTokens);

19.1.2 VotingRightsWithdrawn

Event for logging the voter withdraw voting rights

---

event VotingRightsWithdrawn(address indexed voter, uint256 numTokens);

19.1.3 VoteCommitted

Event for logging the voter commit vote for job hash

---

event VoteCommitted(address indexed voter, uint256 jobID);

19.1.4 VoteRevealed

Event for logging the voter reveal the commit vote

---

event VoteRevealed(address indexed voter, uint256 jobID, bytes32 secretHash);

19.2 Modifiers

19.2.1 isDisputeJob

Check the job hash is the dispute job

```
modifier isDisputeJob(uint256 jobID){
    uint256 jobStatus = bbs.getUint(BBLib.toB32(jobID,'JOB_STATUS'));
    require(jobStatus == 4);
    require(bbs.getAddress(BBLib.toB32(jobID, 'DISPUTE_WINNER'))==address(0x0));
    _;
  }
```

19.3 Functions

19.3.1 isAgaintsPoll

Check this Poll started for the job Hash has againts or not

---

function isAgaintsPoll(uint256 jobID) public constant returns(bool)

Return: (Bool)

19.3.2 setBBOReward

Set bbo reward address, use for send reward to the voter. Only owner can invoke

---

function setBBOReward(address rewardAddress) onlyOwner public

19.3.3 requestVotingRights

The voter request voting rights by lock the number token, each locked token is 1 vote

---

function requestVotingRights(uint256 numTokens) public

19.3.4 withdrawVotingRights

The voter withdraw the locked token

---

function withdrawVotingRights(uint256 numTokens) public

19.3.5 checkBalance

check the locked token balance

---

function checkBalance() public view returns(uint256 tokens)

Return:

19.3.6 commitVote

Voter commit vote for the dispute job

---

function commitVote(uint256 jobID, bytes32 secretHash, uint256 tokens) public isDisputeJob(jobHash)

Modifiers: `isDisputeJob`

19.3.7 revealVote

Voter reveal vote for the dispute job

---

function revealVote(unit256 jobID, address choice, uint salt) public isDisputeJob(jobHash)

Modifiers: `isDisputeJob`

19.3.8 checkHash

Voter can check the `secretHash`

---

function checkHash(uint256 jobID, address choice, uint salt) public view returns(bool)

19.3.9 claimReward

Voter claim reward

---

function claimReward(uint256 jobID) public

19.3.10 calcReward

Calculate the reward of the dispute job hash

---

function calcReward(uint256 jobID) constant public returns(uint256 numReward)

Return: Number of reward.

CHAPTER 20

AdminUpgradeabilityProxy