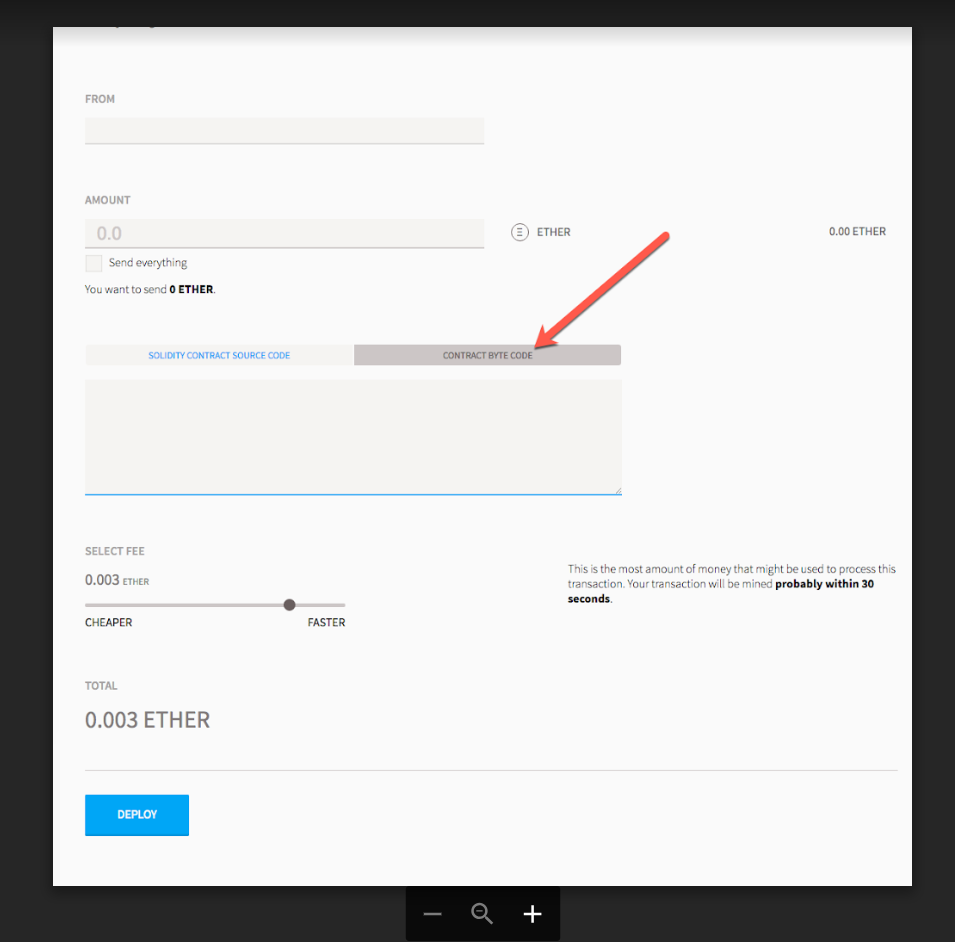
**Toke contract deployment**

After getting the bytecode - instert it into the contract bytecode field (present in Mist/Ethereum wallet)

<https://goo.gl/RygC2P>



4 arguments need to be added as follows:

`000000000000000000000000705ccef62e9c42678b3aa4b73297a3845dfa470b` (we remove 0x in the beginning of the address and replace it with 000000000000000000000000 (24 zeroes)). You should get sth like this:

```{bytecode-from-file}000000000000000000000000705ccef62e9c42678b3aa4b73297a3845dfa470b000000000000000000000000705ccef62e9c42678b3aa4b73297a3845dfa470b000000000000000000000000705ccef62e9c42678b3aa4b73297a3845dfa470b000000000000000000000000705ccef62e9c42678b3aa4b73297a3845dfa470b```

The addresses go as follows, and need to be placed in this order:

- presalePool

- CCTPool

- BCIDeveloperPool

- TreasuryPool

Then proceed with Deploy Contract

After the contract has been deployed - select Watch contract, insert the address of the deployed contract and the abi file contents (also present in the repo).

Crowdsale contract deployment

Identical. There is only 1 argument required though - the address of token contract (with 24 zeroes instead of 0x).

After the crowdsale contract is deployed, transfer all tokens present on the contract creator's account, to crowdsale contract address.

Therefore crowdsale contract will have a balance in tokens that it will be able to use (send to people contributing)

**Here's the general logic for the converter mechanism.**

BC Gateway

Components include:

1. Backend (connecting site)

2. Ethereum Smart Contract (BlockCerts token contract)

3. BlockCerts Smart Contract

Deposit:

Backend is required to monitor every block on Ethereum for transactions to BlockCerts Token smart contract.

`**getBlock**` is used to receive the block

https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethgetblock

A JSON array is returned. It contains a property `transactions` - an array of transaction hashes, of the transactions within the block.

To get info about a particular transaction we user `**getTransactionReceipt**`

https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethgettransactionreceipt

We receive a JSON array that contains a property called `logs`. This is an array of events that were thrown on transaction's execution.

If there are no such events - this transaction is skipped.

if there are such events - we go over them and check each.

The log would look as follows:

{

"address": "0x25ff5573045afd6ef9851cd5847baa0848689d27",

"topics": [

"0xccfaeb3043a96a967dc036ab72e078a9632af809671bc2a1ac30a8043645f89e",

"0x0000000000000000000000005b08e4596b6c21d8190a8e745d6e028d57680bb3",

"0x00000000000000000000000025ff5573045afd6ef9851cd5847baa0848689d27"

],

"data": "0x000000000000000000000000000000000000000000000000000000000000000a",

"blockNumber": 3475614,

"transactionHash": "0xe6bbc4aed5a6152378cb558e9026e66d7d45016a702359afe52c434ea3361f7d",

"transactionIndex": 2,

"blockHash": "0x8bde0d1e5cadff45f7615a7d8e7fead075a0cdbb4ac77b2b8acdd1f09e50fe80",

"logIndex": 0,

"removed": false

}

When going over the log, backend checks `address` property first. If it DOES NOT contain the address of BlockCerts token contract - transaction is skipped. Then comes a check on whether the zero element of `topics` array is `0xccfaeb3043a96a967dc036ab72e078a9632af809671bc2a1ac30a8043645f89e`.

This is a sha3 hash of the event `event Convert(address indexed from, address indexed blockcertsAddress, uint256 \_value);

` from BlockCerts token contract.

If both checks are passed - we get an Eth token-to-BC conversion transaction. The first element (or the zero element) in `topics` array is the address, from which the tokens are transferred, the second (and last) is the address on BCBC, which needs to be credited.

*The topics array contains:*

*event hash*

*token sender address*

*BC address to be credited.*

*I just mentioned two ways of numbering: human (1,2,3) and computer (0,1,2)*

The amount converted is `data` property. It is an integer in `hex` format. It can be converted to a regular number as follows:

`parseInt('0x000000000000000000000000000000000000000000000000000000000000000a', 16)`

Backend therefore gets the info on which BCBC address needs to be credited with how much BC. An according method on BlockCerts gateway contract is called by backend then - specifying the address and the amount - and the BC gateway contract sends a transaction transferring funds from its balance to the destination address.

BC->ETH conversion will follow pretty

A proper implementation will require a contract on BC side - which we will write.

Then we can describe the methods and functions called in it, and the BC->ETH conversion process.

# Ethereum to BCBC after convert function is triggered

## Background Processing

1. step 1 - stage blocks from Ethereum - **Done**
2. step 2 - stage transactions receipts from Ethereum – **Done (fix for failed process)**
3. step 3 - parse transactions receipts to get transfer account and amount
4. step 4 - api support explicit transfer or implicit transfer???
5. step 5 - reporting for the same

# BCBC to Ethereum after convert function is triggered

## Background Processing

1. Not yet started

# KYC Process

Yes, the KYC stuff has not been uploaded to GitLab.

KYC contract is already on the network (it is by default deployed in the genesis block).

Its address is: 0x000000000000000000000000000000000000000a

Admins are the accounts that can change server address. Server is the server that will be sending hashes of the users' inputs during KYC.

userAuthorization function is called by the account that needs to be added as authorized (general user's accounts call it).

There is no owner account per say for the KYC contract, as it was added in the genesis. But the default admin is the same "master account". Further admins can be added/removed by that account.

As for the steps:

1. Hash of a user's KYC data is generated by the server

2. A call of createUserInfo function is made - the hash is added to the contract

3. User account that has been added in createUserInfo needs to call userAuthorization. BCERTS must be sent along with this call

4. isAuthorized function check is made every time a user tries to deploy a contract or interact with an existing contract (excluding INK and KYC)

# BCBC Smart Contract

<https://gitlab.1to1core.com/girish/BlockCerts_ICO/blob/master/contracts/Converter.sol>

Contract needs funds

We've resynced and see the contract owner. What we also see is that the converter.sol contract on the side of BCBC has a balance of 0. It needs balance in BCERTS and INK to send out transactions.

Yes. It needs funds to send to the addresses that need to be credited.

The mint is triggered by the contract owner and the fund will be debited from the contract address... in that case the owner will get's his INK deducted... does the same happen to contract address or only the BCERT will be deducted...

mint function is ownerOnly - therefore it must be called by the contract owner. caller spends ink - therefore owner spends ink.

BCERTS however are sent from the contract - therefore the contract itself needs to have a balance in BCERTS to be sent to users who want to get their BCERTS

Converter.mint -> Final token transfer for the payments made through Ethereum ICO or CC.

Converter.convert -> Converting tokens back to Ethereum.

# Configuration

BCERT token value with respect to dollars.

Ethereum