FCE (FCECHAIN®) BLOCKCHAIN

The Net-Zero “Green Blockchain”
for a Sustainable Economy

CONTENT

CONTENT

01

FCEchain®

The FCE Blockchain is a PoA/**PoT** (Proof of Authority/Proof of Trust) EVM blockchain based on Consensys' Quorum solution. An open-source network client with a GPL license was implemented in the Go language to launch the FCE network.

The FCE network has been launched as an FCE Group's permissioned (private) blockchain.

Architecturally, it is designed so that FCE can always control the minimum number of validators (nodes) necessary to achieve consensus in the network. At the same time, third-party/external validators in the network are involved in creating new blocks in precisely the same way.

At the genesis file creation stage, the FCE network was configured to generate a block every 30 seconds at most. The network capacity is estimated at 10k t\s.

All users (FCE network participants) are disclosed and go through the KYC process for admission to work with the blockchain. This condition is mandatory for all FCEM coin holders and network participants.

To eliminate inefficient competition (from the energy consumption and climate impact perspectives) and because all network participants are disclosed, validators are rewarded sequentially and in turn when processing each block in the network that contains at least one user transaction. The validator's reward mechanism was implemented by harnessing the **RevardValidators** smart contract. It sends one native FCEM coin to the validator from the smart contract, which, as per the FCE Whitepaper, stores the funds allocated for this purpose at the time of the initial emission of the asset. Thus, the tokenomy of the project is deflationary because the initial supply of coins does not increase.

Since the FCE blockchain is permissioned (private), all users interact with it through a single RPC server. The FCE RPC server is implemented to allow only a particular set of valid commands - the basic smart contracts of the FCE blockchain - to enter the network. All direct requests - e.g., a transaction to transfer a native coin - are entirely blocked by the server from the very beginning, including at the stage of the proxy server, where an additional check of the user's KYC status occurs. Thus, all network functionality is implemented through a set of core smart contacts that allow the FCE team to make improvements to the network as a whole.

Diagram #1. Interaction flow between the User and FCE blockchain

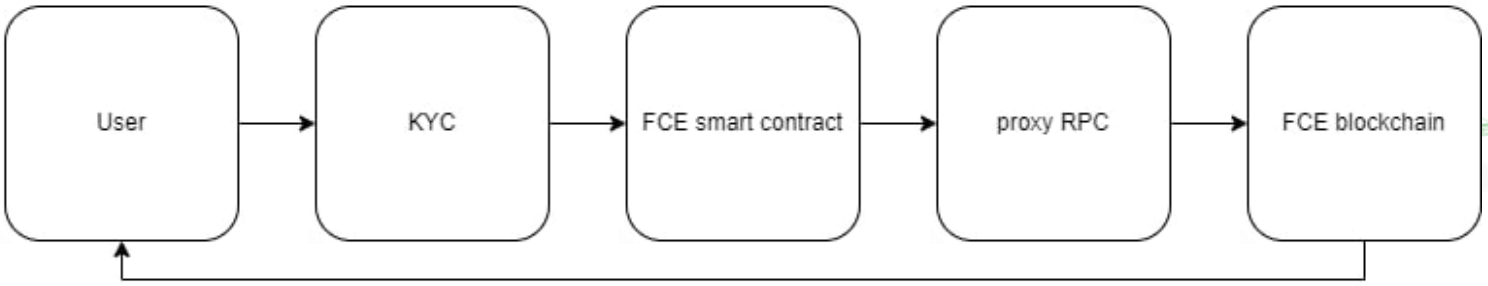


Diagram 1.

The current FCE architecture has made it possible to create a blockchain network with literally a zero-carbon footprint. Since each user transaction is carried out through the invocation of a smart contract, FCE net can instantly redeem its carbon footprint. This mechanism is implemented using tokenized carbon credits provided by FCE partners. The average price of one transaction is calculated in the carbon footprint equivalent. It is considered a commission in the native FCEM currency of the FCE network when a user performs a transaction. This fee offsets the carbon footprint of the transaction by purchasing a tokenized carbon certificate equal to the value of the transaction's footprint. In other words, the user initiating the transaction is obligated to pay a climate impact fee for their transaction. Afterward, the user receives an SBT token (an offset certificate) as a redemption certificate. The design of the token is implemented in the soulbond - token ERC 20 standard without the transfer function (i.e., an unmovable token). This token is a fraction of an offset certificate pre-tokenized by FCE to redeem the network footprint.

Diagram #2. The transaction's carbon footprint offset mechanism

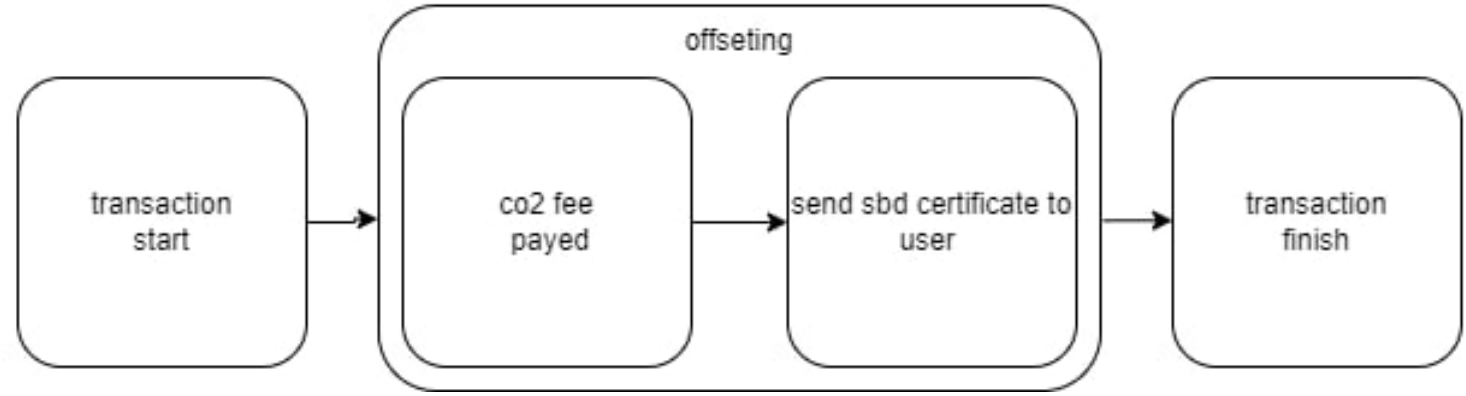


Diagram 2.

A mechanism for registering each address was implemented to simplify the network regulation process. A smart contract must be signed before the user starts interacting with other network contracts. This mechanism makes the entire system much more flexible, allows the assignment of statuses to users, and creates a hierarchical structure of user roles within the network. In other words, the functionality of the FCE blockchain enables the creation of moderators, arbitrators, and other positions within the network with given powers, which opens up a new level of interaction between users to create transparent digital democratic governance.

According to the FCE (former FOODCOIN) whitepaper, when the network was launched, 5,300,000 (5.3B) units of a native coin of the FCEM network were issued. These funds were distributed to smart contract addresses for the funds. Nine percent of the total supply was credited to the smart contract, allowing FOODCOIN holders (since the ICO in 2017) to claim (i.e., migrate their balances) from the FCE private network at a ratio of 1:1. Thus, the mechanism for migration of an asset from a public to a private network was implemented. As the first step, the project tokens were frozen in the Ethereum mainnet in the holders' wallets. Further, the owners' data and balances were placed in a smart contract, through which the users could receive FCEM on their balances.

A cross-chain interaction with the public Ethereum network has been implemented on the bridge principle. The bridge is designed as a standard solution using freeze/unfreeze and mint/burn assets' contracts in private and public networks. Thus, users will be able to withdraw the FCEM network's native coin to the Ethereum mainnet to use in the form of a token. This solution increases the interoperability and liquidity of FCEM coins.

Diagram #3. Bridge

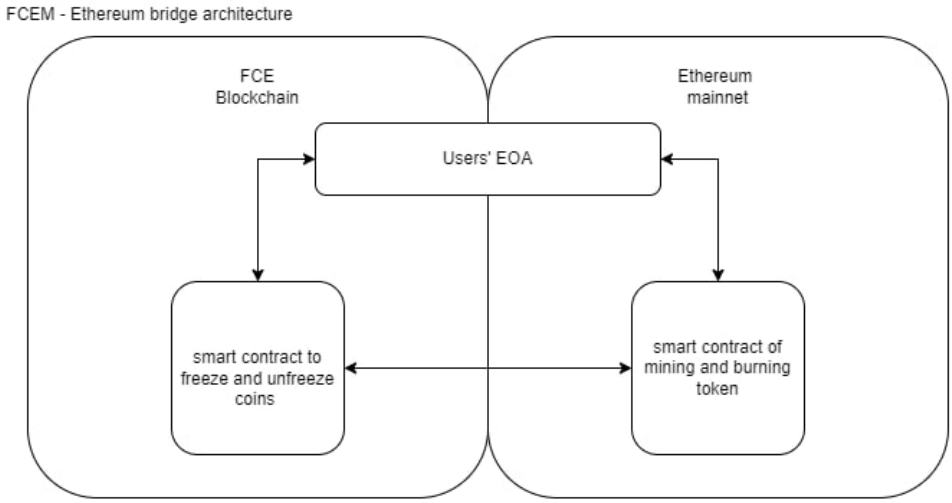
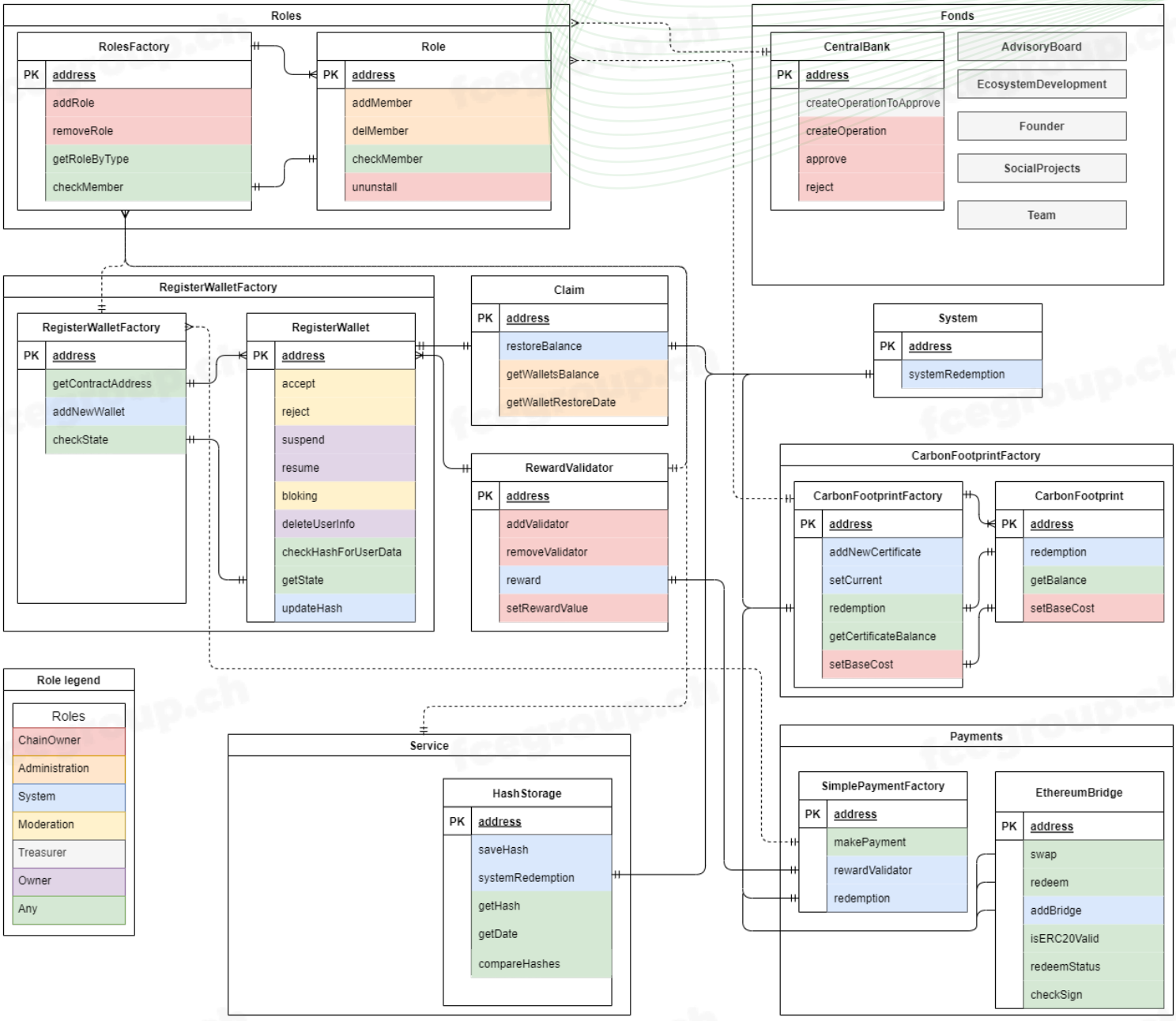


Diagram 3.

Deployed contracts, interdependencies, interoperability, and public functions are described below.

02

Contract dependency diagram.



03

Contracts

3.1. RolesFactory

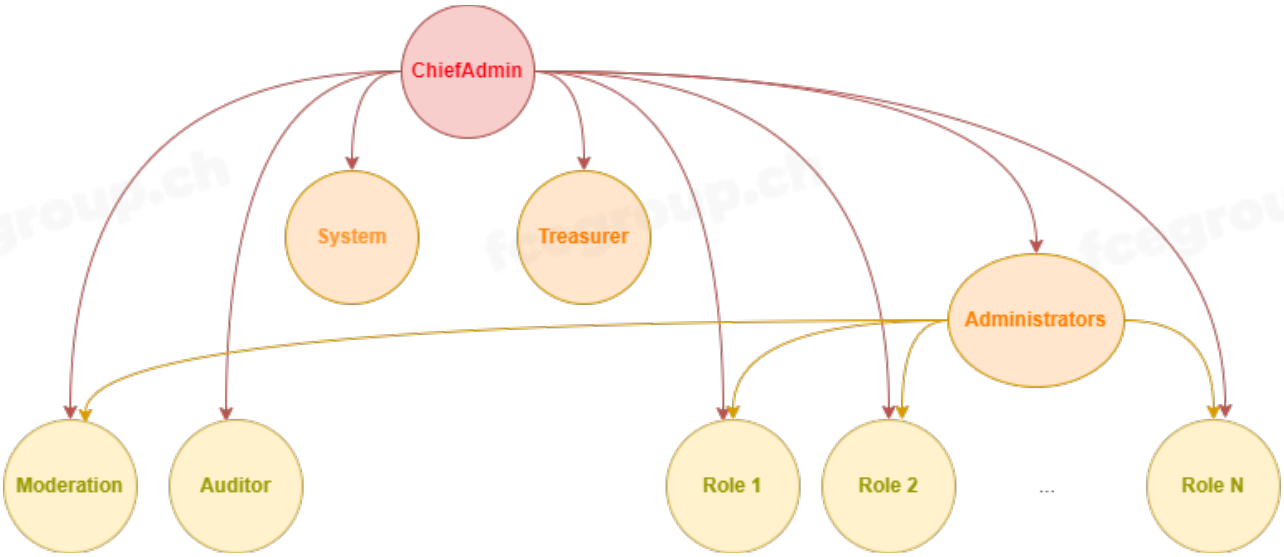
Contract Factory of Roles Model.

A hierarchy system has been implemented: when members of a higher-order role have the right to edit all lower-order roles if they are connected vertically (by direct inheritance).

4 role contracts are pre-deployed:

1. **Administration** - managed by ChiefAdmin, **roleType = 1**
2. **System** - managed by ChiefAdmin, **roleType = 2**
3. **Treasurer** - managed by ChiefAdmin, **roleType = 3**
4. **Moderation** - managed by Administration and ChiefAdmin, **roleType = 4**

The following tree of subordination of the role model is implied. Arrows indicate the ability to manage a role (not only for direct relations but for deeper levels).



3.1.1. Public methods

CheckMember(int **roleType**, address **account**) returns(bool)	Checks if the **account** belongs to a role contract of type **roleType** by calling the **checkMember()** method on the target **Role** contract.
isChiefAdmin() view returns(bool)	Checking the transaction initiator (tx.origin) for it is **ChiefAdmin Role**

3.2. Role

A contract that contains a list of members in a specific role.

3.2.1. Public properties

Name	Type	Description
RoleName	string	Roles name
Type	int	IDs of role
ParentType	int	ID parents role

3.2.2. Public methods

checkMember(address address) view returns **(bool)**	Checking whether the **address** belongs to a role.
checkAdmin(address **account) view** returns**(bool)**	Checks if the **account** belongs to a management role.

3.3. Central Bank

The contract of the Central Bank, where transactions for the transfer of initially placed **FCEM** are made and recorded.

Members with the **Treasurer** role can create operations for confirmation, after which **ChiefAdmin** can decide whether to approve and carry out the operation or reject the operation.

ChiefAdmin also has the right to create and conduct operations without confirmation.

3.3.1. Contract data types

3.3.1.1. Enum TType Operation

- 0: none
- 1: transfer
- 2: receive

3.3.1.2. TOperation

Property name	Type	Description
id	uint128	Operations ID
datetime	uint128	Date/Time of operation in Unix timestamp format
operation	Enum **(TTypeOperation)**	Operation type (ID)
amount	uint256	Transfer amount
recepient	address	Receiver wallet addres / contract address
isApproved	bool	Flag of approval of the operation of **ChiefAdmin**
createdBy	address	Creator wallet address
note	string	Operation description. As a sample - the reason for the reject.

3.3.2. Public properties

1. **Completed operations** - the transfer of funds was carried out according to the data specified in **TOperation.**
2. **Operation being confirmed** - the transfer is under consideration by the ChiefAdmin and has not yet been completed.
3. **Rejected operations** - operations that were rejected by **ChiefAdmin** for confirmation.

Members of the **Treasurer** role can create operations for confirmation, after which **ChiefAdmin** can decide whether to approve and carry out the operation or to reject the operation.

ChiefAdmin also has the right to create and conduct operations without confirmation.

Property name	Type	Description
_CompletedOperations	array**[TOperation]**	The array of completed operations
_OperationToApproved	array**[TOperation]**	Date/Time of operation in Unix timestamp format
_RejectedOperations	array**[TOperation]**	The array of rejected operations.

Also, as per the White Paper, along with the contract of the Central Bank, five contracts of specialized funds have been deployed to make targeted transfers for the specified needs of a particular fund.

Namely:
- **AdvisoryBoard**
- **EcosystemDevelopment**
- **Founder**
- **SocialProjects**
- **Team**

The functionality of these contracts is completely identical to the **CentralBank** contract.

3.4. Register Wallet Factory

A contract factory for linking and managing the state of a wallet address to the TT system.

When registering a wallet, the **RegisterWallet** contract is deployed with the wallet address attached to it. The personal ID and hash from the set of user identification fields (Name, e-mail, date of birth) specified by the user during registration on the **TT** portal are verified through the KYC service.

Through this contract, members of the **Moderation** role and higher have the right to carry out moderation by changing the status of the user's wallet.

Property name	Type	Description
Status	enum	Contains the value for status describing the state of the wallet 0 = Pending 1 = Suspended 2 = Accepted 3 = Rejected 4 = Blocked 5 = Deleted 6 = None

3.4.1. Public methods

Contract factory for linking and management state a wallet address to the TT system.

checkState(address **wallet Address) view** returns (enum **Status**)	Check wallet status **walletAddress**, returns value of **Status** variable
getContractAddress(addr ess **wallet)**	Getting the **wallet** binding contract address
addNewWallet(int **userID,** bytes32 **userHash)**	Creating a new **RegisterWallet** binding contract with saving **userID** and **userHash** on it When creating a contract, it is checked that for this msg.sender wallet does not have a previously created **RegisterWallet** Can executed only be once for one wallet address.

3.5. Register Wallet

The user wallet binding contract stores primary registration information: the registration date, ID, and hash from all user identification fields (Name, e-mail, date of birth) specified by the user during registration on the **TT** portal and verified through the KYC service.

Moderators and members of the role above it can call moderation functions.

3.5.1. Public methods

checkHashForUserData(bytes32 **userInfo) view** returns(bool **isEqual)**	Checking if the **userInfo** data stored in the contract match those passed as a parameter.
getState() view returns**(Status state)**	Returns the **State**

3.6. Carbon Footprint Factory

Factory of certificate contracts for the redemption of the carbon footprint of transactions in the **FCE network**

3.6.1. Public properties

Name	Type	Description
CurrentCertificate	address	Address of the current certificate contract

3.6.2. Public methods

getTokenCost(uint **tokens) view** returns**(uint)**	Returns the amount of **FCEM** that must be paid to redeem the required amount of CO2 tokens with **CaFFee** tokens
redemption(uint256 **tokens) payable**	Carbon offset method for FCEM. Redemption occurs if the amount of paid FCEM matches the number of requested **tokens** based on their cost **BaseCost**. Calls the carbon footprint redemption method in the current certificate, if there are not enough free tokens for the contract, then redeem the maximum of this, and the rest is redeemed from the next certificate contract.
getCertificateBalance()	Get the remainder on the certificate exception.
getCertificateBalance(uint **certificateIndex)**	Return the balance of the remaining **CaFFee** in the certificate indexed by **certificateIndex** in the _Certificates array of certificates.
isCertificate(address **address) view** returns**(bool)**	Checking passed **address** what this one of the certificates in _Certificates arrays.
getCertificateAddress(uint **certificateIndex) view** returns**(address)**	Returns the address of the certificate contract from the _Certificates array by **certificateIndex.**

getNextCertificate() view returns**(address)**	Returns the address of the certificate that is after the current certificate.
getCurrentIndex() view returns(uint **index)**	Returns the **index** of the currently active certificate
getId(uint **certificateIndex) view** returns(string **certificateID)**	Returns the **certificateID** from the _Certificates array by **certificateIndex**.
getURL(uint **certificateIndex) view** returns(string **url)**	Returns the **URL** from the _Certificates array by **certificateIndex.**
getNote(uint **certificateIndex) view** returns(string **note)**	Returns the **note** from the _Certificates array by **certificateIndex.**
getBaseCost(uint **certificateIndex) view** returns(uint **BaseCost)**	Returns the **note** from the _Certificates array by **certificateIndex.**
getTones(uint **certificateIndex) view** returns(uint **tones)**	Returns the **tones** from the _Certificates array by **certificateIndex.**
getBrand(uint **certificateIndex) view** returns(string **brand)**	Returns the **Brand** from the _Certificates array by **certificateIndex.**
getCreationDate(uint **certificateIndex) view** returns(uint **creationDate)**	Returns the **Date** from the _Certificates array by **certificateIndex.**

3.7. Carbon Footprint

The Contract-certificate of a carbon footprint offset (redemption). With each offset (redemption) through the CarbonFootprintFactory contract, non-transferable tokens (SBT) are credited to the user's address.

3.7.1. Public properties

Name	Type	Description
CertificateID	string	Identifier of the tokenized offset certificate
URL	string	URL forwarding to the tokenized offset certificate page
Note	string	Offset certificate description
Brand	string	The offset certificate supplier’s name
InitTokensCount	uint256	CaFFee certificate’s initial balance
freeTokensCount	uint	Amount of tokens available for purchase
BaseCost	uint	FCEM to CaFFee ratio
CO2Tones	uint	Amount of CO2 tons offseted by this certificate
CreationDate	uint	Certificate issuance date in UNIX format
Symbol	string	The name of CaFFee token to display for users in Metamask Wallet

3.7.2. Public methods

getFreeTokensCount() view returns(uint)	Returns the current balance of CaFFee in the contract
totalSupply() view returns(uint)	Returns the total supply of CaFFee in the contract

3.8. System

A contract with an FCEM balance requires redeeming the carbon footprint of system function transactions of other contracts.

3.8.1. Public methods

getBalance() view returns(uint**)**	Returns the current balance of tokens

3.9. Claim

Contract for restoring the balances of FoodCoin holders from the Ethereum network

3.9.1. Public methods

getSelfBalance() view returns(uint **balance)**	Returns the balance that the initiator can restore.
restoreBalance(address account**)**	Restoring the balance of a specific wallet by the amount stored in the comparison (map) **_Wallets** для address. • To check that the address exists • in the comparsion **_Wallets** • To check that that the address exists • in the contracts of wallet connection **RegisterWalletFabrica** and that the connection status is **accepted** After the transition, to set the value to 0 in the mapping, and the date is recorded to the (map) _WalletRestoreDate with the timestamp when the restore operation has been fulfilled.

3.10. Reward Validator

A contract to reward validators on the FCE network.

FCEM are rewarded to validators one by one. For accrual, another contract must call the reward () function.

There are no public variables and methods for users. The functionality of the contract is available only to members of the role System и **ChiefAdmin**.

3.11. Simple Payment Factory

A contract for a basic transfer of FCEM, with the offset of the transaction's carbon footprint and calling of the reward for the validator in **RewardValidator**.

Since direct transfers of FCEM on the network are prohibited, the functions of this contract must be used.

3.11.1. Special contract data types

3.11.1.1. Enum TOperation

The type of operation performed by the user in the contract is indexed from 0 to 4, respectively, according to the description:

- 0 - PaymentGoods
- 1 - ServicePayment
- 2 - Penalty
- 3 - SystemPayment
- 4 - CarbonFootprintPayment

3.11.1.2. Struct Operation

Structure with data fields containing information about the FCEM transfer

Name	Type	Description
operationId	uint	Identifier of the completed operation, assigned based on the value of **_CurrentOperation** (index of the last operation)
amount	uint	The amount of FCEM in wei forwarded to the recipient, excluding the redemption of the carbon footprint
sender	address	The wallet address of the person making the transfer
recipient	address	Recipient's wallet address
isStatusChecked	bool	The condition for the need to check the RegisterWallet status for Accepted for the recipient
description	string	Arbitrary description of the operation

3.11.2. Public methods

Name	Description
makePayment(TOperation **operation,** address **recipient,** bool **checkStatus,** string **description** **)**	The contract method accepts FCEM and allows you to transfer them to the **recipient**, paying for the redemption of the carbon footprint of the transaction, as well as initiating a validator reward **operation** - Operation type index from TOperation (ENUM) **recipient** - Address of the recipient of the transfer **checkStatus** - If equal to 'true', then the transfer will take place only if the recipient's RegisterWallet.State status is Accepted **description** - Some arbitrary description for the translation

3.12. Hash Storage

Value (tuple) storage contracts {**ID, Hash**}

3.12.1. Special contract data types

3.12.1.1. Struct Data

Structure with data fields containing information about the stored value (tuple)

Name	Type	Description
hashedData	string	Pre-encrypted data
date	uint	Date when the encrypted data has been saved

3.12.2. Public methods

data(string **ID)**	Returns the **Data** structure by the passed ID
getHash(string **ID)** returns **string**	Getting **Hash** by **ID**
getDate(string **ID)** returns **uint256**	Getting Hash by **ID**
compareHashes(string **ID,** string **hashedData)** returns **bool**	Checking the correspondence of the transmitted and stored hash by **ID** The check is case-sensitive.

3.13. FCEMBridge - EthereumBridge

Contracts are required to transfer native or wrapped **WFCEM** currency to private from public or to public from private networks.

The **EthereumBridge** contract deploys to the private FCE network and can receive and freeze transferred FCEM and send them when they are transferred from public networks.

FCEMBridge contracts are deployed to public networks and can issue and burn **WFCEM** tokens, depending on whether the tokens are being transferred into or out of the network.

3.13.1. Special contract data types

Struct **bridgeInfo** - the structure stores field that stores information about bridges-contracts in other networks.

Name	Type	Description
chainId	uint	The unique identifier of the network (blockchain).
ERC20Address	address	The address of the WFCEM contract that will create or burn wrapped coins on the public network.
isValid	bool	Data validity flag in the structure fields and activity of this bridge contract.

Enum **Status** - options for values that can take the hashes of the arguments stored on the contract.

Name	Type	Description
Nonexist	0	Default value: The signature either does not exist or has not yet been used.
Undone	1	The signature is registered but not activated. Reactivation is not possible.
Done	2	Signature activated and tokens issued, cannot be reused.

3.13.2. Public properties

Name	Type	Description
ADMIN_ROLE	bytes32 constant	The hash of the name of the role whose members have the right to manage contracts
isContractAvailable	bool	Contract activity flag
bridgeValidator	address	The address of the **WFCEM** contract that can be transferred tokens.
chiefAdmin	address	**ChiefAdmin** address, assigned when the contract is deployed
Bridges	BridgeInfo[ ]	An array of **BridgeInfo** structures storing information about other bridges
redeemStatus	mapping(bytes32 => Status)	The mapping stores the hashes and their Status. The hash is obtained by linking all the arguments passed to the **redeem()** function, except the signature, using the keccak256 hashing algorithm

3.13.3. Public methods

Name	Description
isBridgeValid(uint **chainId,** address **erc20** **) view** returns(bool)	Checking if the network with the requested chainID is active in the array of bridge
checkSign(address **sender,** address **recipient,** uint256 **amount,** uint256 **chainIdfrom,** uint256 **chainIdto,** address **erc20from,** address **erc20to,** uint256 **nonce_,** bytes **signature** **) view** returns**(bool)**	Checks if the bridgeValidator has actually signed the given signature **sender** - sender address **recipient** - recipient's address **amount** - amount of sent currency **chainIdfrom** - identifier of the network from which the transfer was sent **chainIdto** - identifier of the network to which the transfer was sent **erc20from** - address of the wrapped FCEM contract from which the transfer was sent, if it is equal to EthereumBridge, then the transfer was made from the FCE network **erc20to** - address of the wrapped FCEM contract to which the transfer should be sent, if it is equal to EthereumBridge, then the transfer will be made to the FCE network **nonce_** - the value of the transfer counter at the time of the transfer **signature** - the hash signed by the bridgeValidator
hashMessage(bytes**32 message) pure** returns(bytes32)	Appends the required network prefix to the given hash to get the address of the **bridgeValidator**

Name	Description
swap(address **recipient,** address **erc20from,** uint256 **chainIdto,** uint256 **amount,** address **erc20to)** returns(bytes32 **hashToSign)**	Burns the function initiator's existing **WFCEM** tokens and returns a hash of the passed arguments to migrate. All transfer arguments are passed in the SwapInitialized event **recipient** - recipient's address **amount** - amount of sent currency **chainIdto** - identifier of the network to which the transfer will be sent **erc20from** - address of the wrapped **FCEM** contract from which the transfer will be sent, if it is equal to EthereumBridge, then the transfer was made from the FCE network **erc20to** - address of the wrapped **FCEM** contract to which the transfer should be sent, if it is equal to EthereumBridge, then the transfer will be made to the FCE network **hashToSign** - Hashed translation arguments to be signed by the bridgeValidator

Name	Description
redeem(address **sender,** address **recipient,** uint256 **amount,** uint256 **chainIdfrom,** address **erc20from,** address **erc20to,** uint256 **nonce_,** bytes **signature** **)**	When passing valid parameters and signature, it emits **WFCEM** tokens to the address of the **recipient** in the **amount** Based on the passed arguments, a hash is formed, then using the **ecrecover** function and the hash from the **signature**, the signature is validated if it is **bridgeValidator** - signature is valid. Signature can be used only once and only with certain parameters All transfer arguments are passed in the RedeemInitialized event **sender** - sender address **recipient** - recipient's address **amount** - amount of sent currency **chainIdfrom** - identifier of the network from which the transfer was sent **chainIdto** - identifier of the network to which the transfer was sent **erc20from** - address of the wrapped FCEM contract from which the transfer was sent, if it is equal to EthereumBridge, then the transfer was made from the FCE network **erc20to** - address of the wrapped FCEM contract in this network, if it is equal to EthereumBridge, then the transfer will be made in the FCE network **nonce_** - the value of the transfer counter at the time of the transfer **signature** - the hash signed by the bridgeValidator
isAlreadyAdded(uint **chainId,** address **erc20address) view** returns **(**bool **isAdded,** uint **index)**	Checks if a bridge exists in the **Bridges** array: if so, returns true and the index in **Bridges**, if not, then {false; 0}
split(bytes **signature) pure** returns **(**uint8 **v,** bytes32 **r,** bytes32 **s)**	Divides **signature** (signed hash) into 3 signature parameters **v,r,s** on **ECDSA** elliptic curves
getChainID() view returns **(uint)**	Returns the identifier of the network in which the current contract is located

04

WrappedFCEM(ERC20)

The FCE network's wrapped native currency contract for public Ethereum networks. It is a fungible ERC20 token with additional management functionality. WFCEM tokens in public networks can only be issued when an identical amount of FCEM is frozen in the private FCE network through a special FCEMBridge contract.

4.1. Special contract data types

Enum **Permition** - options for values that address can take on the contract.

Name	Type	Description
global	0	Token transfers allowance determined by isTransactionsOn value
forbidden	1	Token transfers forbidden for this address, despite isTransactionsOn value
allowed	2	Token transfers allowed for this address, despite isTransactionsOn value

4.2. Public properties

Name	Type	Description
ADMIN_ROLE	bytes32 constant	The hash of the name of the role whose members have the right to manage contracts
isTransactionsOn	bool	Global token transfers allowance value to all addresses
ChiefAdmin	address	**ChiefAdmin** address, assigned when the contract is deployed
transctionsOnForHolders	mapping(address=> Permition)	The mapping stores addresses and their Permition values
name	string	Name of token
symbol	string	Symbol of token

4.3. Public methods

Name	Description
transationsOnVal(address **accountFrom,** address **accountTo** **) view** returns(bool)	Checks if **accountFrom** and **accountTo** addresses have allowances for token transfer
transfer(address **to,** uint **amount** **)** returns(bool)	Moves **amount** of tokens **to** address to from contract caller. Returns a boolean value indicating whether the operation succeeded.
allowance(address **owner,** address **spender,)** returns(uint)	Returns the remaining number of tokens that **spender** will be allowed to spend on behalf of **owner** through transferFrom function. Returns zero by default. This value changes when approve or transferFrom are called.
approve(address **spender,** uint256 **amount,** **)** returns(bool)	Sets **amount** as the allowance of **spender** over the caller's tokens. Returns a boolean value indicating whether the operation succeeded.
transferFrom(address **from,** address **to,** **uint amount** **)** returns (bool)	Moves **amount** tokens **from** from to **to** using the allowance mechanism. **amount** is then deducted from the caller's allowance. Returns a boolean value indicating whether the operation succeeded.
balanceOf(address **account) view** returns (uint)	Returns the amount of tokens owned by **account**.
totalSupply() view returns **(uint)**	Returns the amount of tokens in existence.

FCE GROUP AG

Switzerland
Techno-Park Luzern CH-6039, Root D4
https://fcegroup.ch
info@fcegroup.ch