Prelude

What's the goal of this handbook?

To slowly start building this up:

- Critical docs: Concepts, Guides and References.
    - Concepts: The internal technical detail, research and relevant topics.
    - Guides: A guide on how the features can be used from a user perspective.
        - User: How to interact and use the node as a regular user.
        - Developer: How to interact on a deeper perspective as a developer.
        - Institution: How to interact with the node for exchanges, dealing with updates, best practices etc.
    - References: A to-the-point reference that's used for quick references.
- Standard practices for coding, git, things like how to add a new data structure, add rpc, etc
- Testing: how to write a test and contribute (so we can welcome external contributions too), and release checks
- Links to pink paper, external resources of all kind (a dedicated page similar to github awesome-{x} repos.
- Anything that's doesn't belong in the source code or is process related.

How do I read and write this handbook?

The book uses the simplest Markdown generator available, `mdbook` - intentionally. No big features. It's simple. Any engineer should be able to learn and use it in minutes - not hours.

- Reading the handbook
- Contributing to the handbook

Learn about it here: mdbook

Attributions

Parts of this handbook were edited from the Bitcoin Wiki.

Concepts

Blocks and the Blockchain

Each DeFiChain block consists of

- a block header: contains metadata about the block and its contents and links to previous block
- a body: contains a list of transactions in that block.

Blocks are identified by their SHA256 hashes.

Block header

The header format of DeFiChain blocks is as follows. The implementation of the block header can be found in the class `CBlockHeader` in file `src/primitives/block.h`.

Name	Data Type	Description
nVersion	int32_t	Used to create and accept soft forks. Usually 536870912. For details on how nVersion is processed and used, read BIP9.
hashPrevBlock	uint256	Double SHA256 hash of previous block header.
hashMerkleRoot	uint256	Double SHA256 hash of merkle root of transaction merkle tree.
nTime	uint32_t	Unix time of when the miner started hashing the header (according to the miner). Must be strictly greater than the median time of the previous 11 blocks.
nBits	uint32_t	Used for PoS mining. An encoded version of the target threshold this block's header hash must be less than or equal to.
~~deprecatedHeight~~*	uint64_t	[***No longer in use**] Block height of the current block. Block height is tracked by `CBlockIndex` instead of block headers themselves, this remains in the block header for compatibility with previous versions of the node. Removal of this field would require a hard fork.
mintedBlocks	uint64_t	Number of blocks this masternode has mined.
stakeModifier	uint256	A stake modifier is a collective source of random entropy for PoS mining. It is equal to `SHA256({previous stake modifier}, {masternode ID}).`

Name	Data Type	Description
sig	vector	Signed digest block header using miner's public key.

Block Body

The block body consists of

- a list of pointers which map to transactions
- a local variable `fChecked` (not shared between nodes) to track if the block has been successfully validated previously

The list of transactions has type `vector<CTransactionRef>`, and each `CTransactionRef` points to a `CTransaction`. The contents of each Transaction as well as the different types of Transactions are detailed in the Transactions document.

External Resources

- Bitcoin Block Reference
- A Decomposition Of The Bitcoin Block Header
- Bitcoin Protocol Documentation
- Blockchain Forking

Transactions

Transactions

- are used to transfer DFI and dTokens
- have inputs and outputs
- difference between inputs and outputs is considered a *transaction fee*
- **not** encrypted
- identified by their SHA256 hashes

Custom Transactions also exist to interact with dApps such as the DeX and Loans.

Transaction Format

Block bodies contain `CTransaction` which have the following structure:

Name	Data Type	Description
nVersion	int32_t	Currently `4`.
vin	vector	List of input transactions.
vout	vector	List of output transactions.
nLockTime	uint32_t	Block height or timestamp when transaction is final
hash	uint256	Hash of transaction without witness data.
m_witness_hash	uint256	Hash of transaction with witness data.

`CTxIn` is an input transaction, and has the following structure:

Name	Data Type	Description
prevout	COutPoint	Hash and index of transaction holding the output to spend.
scriptSig	CScript	Transaction unlocking script.
nSequence	uint32_t	Sequence number. Defaults to 0xFFFFFFFE.
scriptWitness	CScriptWitness	Witness data for this input transaction.

Similar to Bitcoin's script system, a `scriptPubKey` must be able to unlock the respective `scriptSig`.

Each non-coinbase input spends an outpoint (`COutPoint`) from a previous transaction. Coinbase transactions set `prevout` to `null`, and encode the block height in `scriptSig`.

`CTxOut` is the output of the current transaction, and has the following structure:

Name	Data Type	Description
nValue	CAmount	Amount of DFI to spend.
scriptPubKey	CScript	Defines the conditions which must be satisfied to spend this output.
nTokenId	DCT_ID	ID of token transferred. DFI has ID 0.

Coinbase Transaction

Coinbase transactions have the following properties

- it is the first transaction in a block
- there can only be one coinbase transaction per block
- creates new DFI in mining masternode's wallet

The base block reward is split into 6 different recipients. Recipients marked with UTXO are the only outputs present in the coinbase transaction.

Recipient	Coinbase allocation	Description
Masternode (UTXO)	33.33%	Masternode rewards. Masternode also receives additional income from transaction fees.
Community Fund (UTXO)	4.91%	Community fund controlled by the Foundation, used to finance DeFiChain initiatives. ref
Anchor	0.02%	Fund to pay the Bitcoin transaction fee for anchoring.
Liquidity Pools	25.45%	Distributed to LP providers as incentive.
Loans	24.68%	Incentives for atomic swap and DeFiChain futures.
Options	9.88%	Distributed to options holders. Not in use currently.
Unallocated	1.73%	N/A

External Resources

- Bitcoin Script

- scriptPubKey & scriptSig Explained

Custom Transactions

Custom Transactions are special transactions to interact with dApps on DeFiChain, such as Vaults and the DeX.

Custom transaction can be identified by the first `vout` of a transaction

- begin with a `OP_RETURN`
- followed by a `DfTx` marker (serialized in UTF as `44665478`)
- followed by one character marker denoting the type of transaction (serialized in UTF), conversion table here
- followed by custom transaction data, if required

For example, in the following custom transaction

```
"vout": [
{
    "value": 0.00000000,
    "n": 0,
    "scriptPubKey": {
    "asm": "OP_RETURN 446654784301d823ad3976574a50001f4ca5a5326a1c1232a0a70000",
    "hex": "6a054466547841",
    "type": "nulldata"
    },
    "tokenId": 0
},
...
]
```

The custom transaction can be broken down to

asm	Meaning
44665478	DfTx marker
43	`C` in UTF, corresponds to CreateMasternode message (CCreateMasterNodeMessage)
01	`operatorType` variable
d823ad3976574a50001f4ca5a5326a1c1232a0a7	`operatorAuthAddress` variable
0000	`timelock` variable

Transaction Types

Masternodes

CreateMasterNode

Creates a new masternode in the network

Marker: `C`

Message Format

Variable	Data Type	Description
operatorType	char	1 for P2PKH, 4 for P2WPKH
operatorAuthAddress	CKeyID	DeFiChain address of masternode operator
timelock	uint16_t	minimum time that masternode must be active, either 0, 5 or 10 years

ResignMasternode

Removes enabled or pre-enabled masternode from network.

Marker: `R`

Message Format

Variable	Data Type	Description
nodeID	uint256	ID of masternode to resign

SetForcedRewardAddress

Sets reward address for masternode.

Marker: `F`

Variable	Data Type	Description
nodeID	uint256	masternode ID
rewardAddressType	char	1 for P2PKH, 4 for P2WPKH
rewardAddress	CKeyID	new DeFiChain address for rewards

RemForcedRewardAddress

Removes reward address for masternode if any.

Marker: `f`

Variable	Data Type	Description
nodeID	uint256	masternode ID

UpdateMasternode

Updates masternode metadata.

Marker: `m`

Message Format

Variable	Data Type	Description
operatorType	char	1 for P2PKH, 4 for P2WPKH
operatorAuthAddress	CKeyID	DeFiChain address of masternode operator
timelock	uint16_t	minimum time that masternode must be active, either 0, 5 or 10 years

Custom Tokens

CreateToken

Creates new custom token.

Marker: `T`

Message Format

Variable	Data Type	Description
token	CToken	token definition

Class `CToken` has the following structure

Variable	Data Type	Description
symbol	string	token ticker
name	string	name of token

Variable	Data Type	Description
decimal	uint8_t	how divisible a token can be, fixed to `8`
limit	CAmount	deprecated
flags	uint8_t	sets options for token

The following flags are available for a token:

Bit index	Flag
0	None
1	Mintable
2	Tradeable
3	DeFi Asset Token (tokens backed by collateral, e.g. dBTC)
4	Liquidity Pool Share
5	Finalized
6	Loan Token
7	Unused

Tokens are mintable and tradeable by default.

MintToken

Mints new custom tokens.

Marker: `M`

Message Format

Variable	Data Type	Description
balances	CBalances	amount and type of token to mint

UpdateToken

Set DAT flag for a token to true or false. **Deprecated.**

Marker: `N`

Variable	Data Type	Description
tokenTx	uint256	CreateToken transaction for a custom token
isDAT	bool	is token a DAT

UpdateTokenAny

Update a token, supports all fields in CToken.

Marker: `n`

Variable	Data Type	Description
balances	CBalances	amount and type of token to mint

DeX

CreatePoolPair

Create new liquidity pool.

Marker: `p`

Message Format

Variable	Data Type	Description
poolPair	CPoolPairMessage	metadata for liquidity pool
pairSymbol	string	name of liquidity pool
rewards	CBalances	percentage of coinbase liquidity rewards allocated to pool

`CPoolPairMessage` has the following message structure:

Variable	Data Type	Description
idTokenA	DCT_ID	token ID for first token in LP
idTokenB	string	token ID for second token in LP
commission	CAmount	define fees for swap
ownerAddress	CScript	owner of DeX pool
status	bool	pool status (active/inactive)

UpdatePoolPair

Update a liquidity pool.

Marker: `u`

Message Format

Variable	Data Type	Description
poolId	DCT_ID	pool ID
status	bool	set pool status (active/inactive)
commission	CAmount	define fees for swap
ownerAddress	CScript	owner of DeX pool
rewards	CBalances	percentage of coinbase liquidity rewards allocated to pool P2WPKH

PoolSwap

Swap tokens using a liqidity pool. **Deprecated.**

Marker: `s`

Message Format

Variable	Data Type	Description
from	CScript	transaction input
to	CScript	transaction output
idTokenFrom	DCT_ID	input token ID
idTokenTo	DCT_ID	output token ID
amountFrom	CAmount	amount of tokens to swap
maxPrice	PoolPrice	maximum possible price

PoolSwapV2

Swap tokens using a liqidity pool. Has 16 digits precision instead of 8 in V1.

Marker: `i`

Message Format

Variable	Data Type	Description
swapInfo	CPoolSwapMessage	swap metadata (contents of V1 message)
poolIDs	vector<DCT_ID>	IDs of pools to swap with

AddPoolLiquidity

Mint LP tokens and add liquidity to a liquidity pool.

Marker: `l`

Message Format

Variable	Data Type	Description
from	CAccounts	dToken input
shareAddress	CScript	address for LP rewards

RemovePoolLiquidity

Redeem LP tokens and withdraw liquidity from a liquidity pool.

Marker: `r`

Message Format

Variable	Data Type	Description
from	CAccounts	LP token input
amount	CTokenAmount	withdrawal amount

Accounts

UtxosToAccount

Spend UTXOs from a wallet and transfer tokens to to any DeFiChain address.

Marker: `u`

Message Format

Variable	Data Type	Description
to	CAccounts	account and amount of tokens to send

AccountToUtxos

Creates UTXO from a wallet account.

Marker: `b`

Message Format

Variable	Data Type	Description
from	CScript	UTXOs to spend
balances	CBalances	amount of DFI to send
mintingOutputsStart	uint32_t	number of transaction outputs

AccountToAccount

Transfer tokens between a wallet and any DeFiChain address.

Marker: `B`

Message Format

Variable	Data Type	Description
from	CScript	UTXOs to spend
to	CBalances	account to send tokens to

AnyAccountsToAccounts

Transfer tokens from any DeFiChain address to any DeFiChain address.

Marker: `a`

Message Format

Variable	Data Type	Description
from	CScript	UTXOs to spend
to	CBalances	account to send tokens to

SmartContract

Interact with DFIP2201 contract.

Marker: `K`

Message Format

Variable	Data Type	Description
name	string	name of the smart contract
accounts	CAccounts	account and amount of tokens to send

DFIP2203

Interact with DFIP2203 (Futures) contract.

Marker: `Q`

Message Format

Variable	Data Type	Description
owner	CScript	address to fund contract and recieve token
source	CTokenAmount	amount of tokens to fund future contract
destination	uint32_t	set underlying asset (if source id DUSD)
withdraw	bool	withdraw asset to owner address

Governance

SetGovVariable

Update governance variables.

Marker: `G`

Message Format

Variable	Data Type	Description
govs	map<shared_ptr>	list of variables and values to update

SetGovVariableHeight

Set one governance variable at a particular block height.

Marker: `j`

Message Format

Variable	Data Type	Description
govs	shared_ptr	variable name and value to update
startHeight	uint32_t	height at which change is in effect

Authorisation

AutoAuthPrep

This is an empty custom transaction.

Marker: `A`

Oracles

AppointOracle

Create new oracle at an address.

Marker: `o`

Message Format

Variable	Data Type	Description
oracleAddress	CScript	address of new oracle
weightage	uint8_t	oracle weightage
availablePairs	set	set of token prices available

RemoveOracleAppoint

Remove existing oracle.

Marker: `h`

Message Format

Variable	Data Type	Description
oracleId	COracleId	oracle ID

UpdateOracleAppoint

Update existing oracle.

Marker: `t`

Message Format

Variable	Data Type	Description
oracleId	COracleId	oracle ID
newOracleAppoint	CAppointOracleMessage	new oracle metadata (contents of AppointOracle message)

SetOracleData

Update oracle prices.

Marker: `y`

Message Format

Variable	Data Type	Description
oracleId	COracleId	oracle ID
timestamp	int64_t	timestamp of oracle data
prices	CTokenPrices	array of price and token strings

Interchain Exchange (ICX)

ICX is currently disabled on the mainnet.

ICXCreateOrder

Create ICX order.

Marker: `1`

Message Format

Variable	Data Type	Description
orderType	uint8_t	is maker buying or selling DFI
idToken	DCT_ID	token ID on DeFiChain
ownerAddress	CScript	address for funding DeFiChain transaction fees
receivePubkey	CPubKey	public key which can claim external HTLC
amountFrom	CAmount	amount of asset to be sold
amountToFill	CAmount	amount of tokens in order available to fill
orderPrice	CAmount	quoted price of asset

Variable	Data Type	Description
expiry	uint32_t	block height for order expiry

ICXMakeOffer

Make an offer for an ICX order.

Marker: `2`

Message Format

Variable	Data Type	Description
orderTx	uint256	transaction ID of order
amount	CAmount	amount of asset to swap
ownerAddress	CScript	address for funding DeFiChain transaction fees
receivePubkey	CPubKey	public key which can claim external HTLC
expiry	uint32_t	block height for order expiry
takerFee	CAmount	taker fee

ICXSubmitDFCHTLC

Submit a DFI hash time lock contract (HTLC) for offer.

Marker: `3`

Message Format

Variable	Data Type	Description
offerTx	uint256	transaction ID of offer
amount	CAmount	amount to put in HTLC
hash	uint256	hash of seed used for the hash lock
timeout	uint32_t	timeout for expiration of HTLC in DeFiChain blocks

ICXSubmitEXTHTLC

Submit a BTC hash time lock contract (HTLC) for offer.

Marker: `4`

Message Format

Variable	Data Type	Description
offerTx	uint256	transaction ID of offer
amount	CAmount	amount to put in HTLC
hash	uint256	hash of seed used for the hash lock
htlcscriptAddress	string	script address of external htlc
ownerPubkey	CPubKey	refund address in case of HTLC timeout
timeout	uint32_t	timeout for expiration of HTLC in DeFiChain blocks

ICXClaimDFCHTLC

Claim a DFI HTLC.

Marker: `5`

Message Format

Variable	Data Type	Description
dfchtlcTx	uint256	transaction ID of DeFiChain HTLC
seed	vector	secret seed for claiming HTLC

ICXCloseOrder

Close an ICX order.

Marker: `6`

Message Format

Variable	Data Type	Description
orderTx	uint256	transaction ID of ICX order

ICXCloseOffer

Close an ICX offer.

Marker: `7`

Message Format

Variable	Data Type	Description
offerTx	uint256	transaction ID of ICX offer

Loans

SetLoanCollateralToken

Sets what tokens are allowed as collateral.

Marker: `c`

Message Format

Variable	Data Type	Description
idToken	DCT_ID	token ID
factor	CAmount	minimum collateralization ratio
fixedIntervalPriceId	CTokenCurrencyPair	oracle price for current loan period
activateAfterBlock	uint32_t	block number at which change will be activated

SetLoanToken

Sets loan token.

Marker: `g`

Message Format

Variable	Data Type	Description
symbol	string	ticker for loan token
name	string	name of loan token
fixedIntervalPriceId	CTokenCurrencyPair	oracle price for current loan period
mintable	bool	token's mintable property, defaults to true
interest	CAmount	interest rate for the token

UpdateLoanToken

Update loan token.

Marker: `x`

Message Format

Variable	Data Type	Description
tokenTx	uint256	token creation tx
symbol	string	ticker for loan token
name	string	name of loan token
fixedIntervalPriceId	CTokenCurrencyPair	oracle price for current loan period
mintable	bool	token's mintable property, defaults to true
interest	CAmount	interest rate for the token

LoanScheme

Create a loan scheme.

Marker: `L`

Message Format

Variable	Data Type	Description
identifier	string	loan scheme identifier
updateHeight	uint64_t	block height to create scheme at
ratio	uint32_t	minimum collateralization ratio
rate	CAmount	interest rate

DefaultLoanScheme

Set default loan scheme.

Marker: `d`

Message Format

Variable	Data Type	Description
identifier	string	loan scheme identifier

DestroyLoanScheme

Delete existing loan scheme.

Marker: `D`

Message Format

Variable	Data Type	Description
identifier	string	loan scheme identifier
destroyHeight	uint64_t	block height to destroy scheme at

Vault

Create new vault.

Marker: `V`

Message Format

Variable	Data Type	Description
ownerAddress	CScript	owner of vault
schemeId	string	loan scheme to use

CloseVault

Close existing vault.

Marker: `e`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
to	CScript	address to send collateral to

UpdateVault

Update existing open vault.

Marker: `v`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
ownerAddress	CScript	owner of vault
schemeId	string	loan scheme to use

DepositToVault

Deposit collateral in a vault.

Marker: `S`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
from	CScript	address funding the vault
amount	CTokenAmount	amount of tokens to deposit

WithdrawFromVault

Withdraw collateral in a vault.

Marker: `J`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
to	CScript	address to send collateral to
amount	CTokenAmount	amount of tokens to deposit

TakeLoan

Mint loan token against a loan vault.

Marker: `X`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
to	CScript	address to send loan tokens to
amount	CBalances	amount of loan token to mint

PaybackLoan

Repay an existing loan. **Deprecated**.

Marker: `H`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
from	CScript	address repaying the loan
amount	CBalances	amount of loan token to repay

PaybackLoanV2

Repay an existing loan.

Marker: `k`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
from	CScript	address repaying the loan
loans	map<DCT_ID, CBalances>	amount of loan token to repay

AuctionBid

Bid for an auction on a vault.

Marker: `I`

Message Format

Variable	Data Type	Description
vaultId	CVaultId	ID of vault
index	uint32_t	auction index
from	CScript	address to recieve vault collateral
amount	CTokenAmount	bid amount

FutureSwapExecution

Unimplemented

Marker: `q`

FutureSwapRefund

Unimplemented

Marker: `w`

Protocol and Network

DeFiChain Pink Paper

This is a working version of the Pink Paper for DeFiChain.

Comments and pull requests are welcomed.

Goals

1. Lay out the roadmap for DeFiChain for the next few quarters, esp. 2021/2022.
2. To form the basis of an open discussion with DeFiChain community on the direction of the project.
3. Provide a general product and design direction for the engineering team.

Sections

#	Topic	Remarks
1	Introduction	This document.
2	Emission rate	Under implementation. Targeting May 2021 upgrade.
3	Governance	Also community development fund management. Under implementation. Targeting May 2021 upgrade.
4	Price oracle	Under implementation. Targeting May 2021 upgrade.
5	Interchain exchange (ICX)	Also Bitcoin atomic swap. Under implementation. Targeting May 2021 upgrade.
6	Fees	Draft. Partial implementation.
7	Futures	Early draft.
8	Operator	Early draft.
9	Loan & decentralized tokenization	Early draft.

#	Topic	Remarks
10	Automated options	Early draft.
11	Non-Fungible Token (NFT)	Early draft.

See also

- Presentation deck on 2021-03-30

Disclaimers

1. Content may be updated without notice.
2. Content does not guarantee implementation and does not imply implementation order.
3. Actual implementation may differ from the paper.
4. This my no means signify acceptance by the DeFiChain users and masternode owners.

DFI emission rate

Current

From white-paper:

> DeFiChain is initially launched with a 200 DFI block reward, of which 10% goes to the community fund. The Foundation pledges to guarantee this 200 DFI block reward for at least 1,050,000 blocks since the the first genesis block, so approximately 1 year.

The white paper lays out the expectation for the first year that the block reward is flat at 200 DFI.

2 DFIPs have since been made and implemented that affected the allocation of the 200 DFI, while the total emission remains consistent at 200 DFI.

- DFIP #1: Bitcoin anchor reward source and mechanics adjustments where 0.1 DFI is reduced from community fund to cater for anchor reward
- DFIP #2: DeFi incentive funding where the original 180 DFI per block mining reward is reduced to 135 DFI

To keep the total supply of DFI capped at 1.2 billion, the emission rate of DFI cannot stay consistent forever and will have to be reduced over time.

Foundation and Community Fund

To further decentralize the governance of DeFiChain, the Foundation's role must be diminished significantly, and should not be holding any DFI. As such, it is proposed for the Foundation tokens to be burned entirely so no individuals or entities hold the private keys to the Foundation's DFI.

Community Fund today accumulates at 19.9 DFI per block and requires community funding proposal (CFP) to be made and votes to be called for the fund to be *manually* released.

Community Fund should be removed from address with private key to consensus-managed on-chain voting.

Emission rate

Goals

1. Maintain the same emission rate on the first cycle.
2. Avoid the dramatic "halving" like Bitcoin's.
3. Maintain the 1.2 billion DFI cap.
4. Maintain a steady expected emission rate for foreseeable future.

Current rate

The current emission rate of DFI is 258.1 DFI per block – some of it are funded by airdrop fund.

Description	DFI per block
Mining reward	135
Community fund	19.9
Anchor reward	0.1
DeFi incentives	
BTC-DFI	80
ETH-DFI	15
USDT-DFI	5
LTC-DFI	2
BCH-DFI	1
DOGE-DFI	0.1
TOTAL	258.1

The same rate will be retained on the first cycle.

On top of the above, the following will also be pre-allocated on the first cycle:

Description	In use today?	DFI per block	Proportion
Mining reward	Yes	135	33.33% (guaranteed)
Community fund	Yes	19.9	4.91%
Anchor reward	Yes	0.1	0.02%
DEX liquidity mining	Yes	103.1	25.45%

Description	In use today?	DFI per block	Proportion
Atomic swap incentives	No	50	12.34%
Futures incentives	No	50	12.34%
Options incentives	No	40	9.88%
Unallocated incentives	No	6.94	1.71%
TOTAL		405.04	100%

Cycle & Reduction

Emission rate will start at 405.04 DFI on the first cycle so that it flows smoothly with no change before the implementation. The increment on the first cycle is coming from the destruction of Foundation-owned DFI.

Cycle period is 32,690 blocks, at 37 seconds per block, each cycle takes approximately 2 weeks.

Every cycle, the block reward reduces by 1.658%. The splitting of the rewards for respective incentives will be maintained at the same proportion as laid out on the schedule above.

Unused reward will be burned. Burned rewards will not be re-introduced, therefore counted towards the 1.2 billion supply cap.

Rewards proposed may be adjusted with future on-chain governance changes, however, masternode (mining) rewards are guaranteed to be 33.33% of the total block reward.

Fees on DeFiChain

Usage of DeFiChain generally consists of the following fees:

1. Miner fee
2. Operator fee
3. DeFi fee

Miner fee

Miner fee is the small fee, paid in UTXO DFI, to masternodes (also referred to as miners) for transaction validation and inclusion in a block.

This works the same as Bitcoin blockchain, measured in data size, e.g. price per byte.

Operator fee

As part of DeFiChain generalization, many services will soon be run by operators.

For providing of services and applications on DeFiChain, operators are able to determine fees for the services that they would charge freely. Fees can be in any form of tokens, DFI or operators' own tokens, or a combination of multiple tokens. What an operator chooses to do with the operator fee is at the operator's discretion.

DeFi fee

DeFi fee is charged for all DeFi transactions on DeFiChain regardless of operators.

Fee schedule

All fees are burned in a trackable and transparent manner.

The following fees are for illustrative purposes only and have yet to be fully determined.

Operations	Burn amount (in DFI)
Masternode	
Registration	10

Operations	Burn amount (in DFI)
Operator	
Registration	1000
Tokenization	
Create a new token	100
Liquidity pool	
Create pool	300
Swap (for non-DFI pairs)	0.01
Governance	
Initiate community fund request	5
Initiate vote of confidence	25
Initiate block reward reallocation proposal	250
Interchain exchange	
Atomic swap	0.01
Oracle	
Create price feed	100
Appoint oracle	20
Loan	
Loan interest	TBD
Liquidation fee	TBD
Option pool	
Create pool	300
Option contract creation	0.01
Bonus payout fee	TBD
Future	
Create future market	300
Trade fee	TBD

Futures

Stocks, commodities and real-world tokenization will be offered on DeFiChain via futures contract, offered by Operators.

Future is offered by operators with the assistance of price oracles, allowing prices of assets to be tracked, real world or otherwise.

DeFiChain will support 2 types of futures:

1. Fixed expiry

- With a fixed expiry at a certain date in the future.
- At the time of expiry,

2. Perpetual swap

- With funding mechanics

Mechanics

1. Margin contract

- Trader is able to fund their Future trading position through this margin contract.
- Contract can be funded with DFI and other tokens that operators decide, e.g. BTC.
- Operator can also define *initial margin* and *maintenance margin* for a margin contract.

2. Asset

- Operator decide assets to list.
- Assets can be stocks, commodities or anything that has a price. It is operator's responsibility to ensure that there are good price oracles for traded assets.

3. Order

- User is able to make an order. It can be a **long** or **short** order.
- Orders are matched automatically on-chain.

4. Settlement

- Fixed expiry future will be settled automatically based on settlement oracle price published.
- Perpetual swap future has no settlement. There is instead a **funding rate mechanism**.

5. Perpetual swap funding

- Funding rate is automatically determined by DeFiChain based on fixed algorithm: oracle price, perpetual swap trading price.
- Funding period can be set by operator, but it should be between 1x to a max of 12x per day. For reference, most centralized perpetual swap offerings have a 8h funding interval (3x per day).

6. Liquidation

- When net margin (total margin + unrealized PnL) is less than maintenance margin, forced liquidation would occur.
- TBD: Either automated, operator-run, or community-run with incentives.

Enahncements

1. Future can further be enhanced to allow leverage. DeFiChain will start with no leverage (1x) and will introduce leverage once the mechanisms are stable.

RPC

TK

On-chain Governance

Governance summary

Voting is carried out every 130,000 blocks, roughly once every calendar month.

Types of governance:

1. Community development fund request proposal

    - Requesting of community development fund.
    - Requires simple majority (> 50%) by non-neutral voting masternodes to pass.
    - Fee: 10 DFI per proposal

2. Block reward reallocation proposal

    - Note the miner's reward (PoS) are guaranteed and cannot be reallocated.
    - Requires super majority (> 66.67%) by non-neutral voting masternodes to pass.
    - Fee: 500 DFI per proposal

3. Vote of confidence

    - General directional vote, not consensus enforceable.
    - Requires super majority (> 66.67%) by non-neutral voting masternodes to pass.
    - Fee: 50 DFI per proposal

All fees are trackably burned.

Community fund request proposal (CFP)

Current

Currently every block creates 19.9 DFI to community fund address that is controlled by Foundation.

Improvements

The amount at `dZcHjYhKtEM88TtZLjp314H2xZjkztXtRc` will be moved to a community balance, we call this the "Community Development Fund".

This fund, will be similar to Incentive Funding and Anchor Reward, they have no private keys and are unrealized by default unless allocated.

Voting is carried our every 90,000 blocks (roughly once every calendar month). This should be configurable as chainparams and requires a hardfork to update. The block where the vote is finalized is also known as "Voting Finalizing Block".

User is able to submit a fund request that is to be voted by masternodes. Each fund request costs non-refundable 10 DFI. The 10 DFI is not burned, but being added to the proposal for that will be paid out to all voting masternodes to encourage participation. Each fund request can be also specify `cycle`. Cycle is defaulted to `1` for one-off fund request, this also allows for periodic fund requests where a request is for a fixed amount to be paid out every cycle (every 90,000 blocks) for the specified cycles if the funding request remains APPROVED state.

Data structure of a funding request

User submitted data

1. Title (required, this may also be implemented as hash of title to conserve on-chain storage)
2. Payout address (required) this can be different from requester's address
3. Finalize after block (optional, default=curr block + (blocks per cycle / 2))
    - Max valid value would be curr block + 3 * blocks per cycle. No minimum.
    - The rationale for this value is to allow sufficient time for requests to be discussed and voted on, esp. when it is submitted closer to the end of current voting cycle.
4. Amount per period (in DFI)
5. Period requested (required, integer, default=1)

Internal data

1. Completed payment periods (integer)
2. State
3. Voting data

States of funding request

Funding requests can have one of the following states

1. `VOTING`: Successfully submitted request, currently undergoing voting state
2. `REJECTED`: At voting finalizing block, if a request does not achieve. This is a FINAL state and cannot be re-opened.
3. `COMPLETED`: A funding request that has been paid out entirely. When completed payment periods is equals to period requested, request is marked as `COMPLETED`. . This

is a FINAL state and cannot be re-opened.

The absence of the state `APPROVED` is intended. An approved request will simply have its payout made and increment the completed payment period. If payment has been completed fully, request state will be changed to `COMPLETED`, which is a final state. A corollary to this allows for a request that requested for a payout of 1000 DFI every month and has received for 2 months to be rejected at the 3rd month if there are more no votes to yes votes.

RPC

1. `createdovcfr '{DATA}'`
    - `DATA` is serialized JSON with the following:
        - `title`
        - `finalizeAfter`: Defaulted to current block height + 90000/2. All eligible proposals have to be submitted at least 90k blocks before the finalizing block and not more than 270k blocks before.
        - `cycles`: Defaulted to `1` if unspecified.
        - `amount`: in DFI.
        - `payoutAddress`: DFI address
    - 10 DFI is paid along with the submission.
    - Request is finalized after submission, no edits are possible thereafter.
    - Returns `txid`. The `txid` is also used as the proposal ID.
2. `vote PROPOSALID DECISION`
    - `PROPOSALID` is the `txid` of the submitted funding request.
    - `DECISION` can be either of the following:
        - `yes`
        - `no`
        - `neutral`
    - A masternode is able to change the vote decisions before the request is finalized, the last vote as seen on the block BEFORE finalizing block is counted.

Votes

A masternode is deemed to be eligible to cast a vote if ALL the following conditions are satisfied:

1. The masternode is in `ENABLED` state during vote submission AND at vote finalizing block.

2. The masternode has successfully minted at least 1 block during vote submission AND at vote finalizing block.

Any eligible masternodes can cast a vote to any proposal that is in `VOTING` state, regardless of completed payment period.

Masternodes are able to cast any of the following votes:

1. YES
2. NO
3. NEUTRAL

At every vote finalizing block, a community fund proposal (CFP) is deemed to be eligible to be paid out of all the followings are true:

1. There are more YES votes than NO votes. Simple majority wins. NEUTRAL votes are not counted. If there are equal amount of YES and NO votes, the CFR is deemed to have failed and rejected.
2. The voting masternodes eligibility must be `ENABLED` at the finalizing block and have mined at least 1 block.
3. Quorum: All proposals require at least 1% of voting masternodes of all active masternodes that mined at least 1 block before the finalizing block.

Other proposals

Other proposals such as block reallocation proposal and vote of confidence will follow the same voting cycle.

Supplementary website

While blockchain's as data storage is desired to maintain impartiality and anonymity of discussions, it is not a conducive medium for a wider discussion. A supplementary website will be made available to facilitate all proposal discussions and track the progress.

A good supplementary website is suggested to carry the following features:

1. **Proposer identification** to facilitate meaningful discussion among the community and the proposer. This does not, however, imply the need of a user account system.
2. **Proposal extension and title hash validation**. As the blockchain only contains the hash of title, the supplementary website must allow the proposer to enter the full proposal title hash and enter any additional descriptions on the proposals.
3. **Vote tracking**. Ability to track votes as it happens.

An example good supplementary website would be https://www.dashcentral.org/budget for Dash.

Interchain Exchange

Goals

Allow users to trustlessly and seamlessly swap between DST and Bitcoin BTC. BTC in the rest of this document will only referring to BTC on Bitcoin blockchain and not BTC DST.

Key areas of focus, in decreasing order of importance:

1. Safety
2. User experience – it should be easily presentable on a client UI
3. Seamlessness - most of the steps should be automated at client

Components

1. Hash Time Lock Contract
    - A.k.a. HTLC
    - Part of Bitcoin script, and DFC side, it's a new construct for DST.
    - Requires both participants to be active.
    - H = Hash(S)
2. Inter-chain Exchange
    - A.k.a. ICX / XCX
    - This is on-chain (DeFiChain) trustless exchange with "traditional" orderbook
3. SPV wallet
    - Simple Payment Verification Bitcoin wallet
    - Distributed as part of DeFiChain node
    - This is why we are only supporting Bitcoin, at least for now.

Glossary

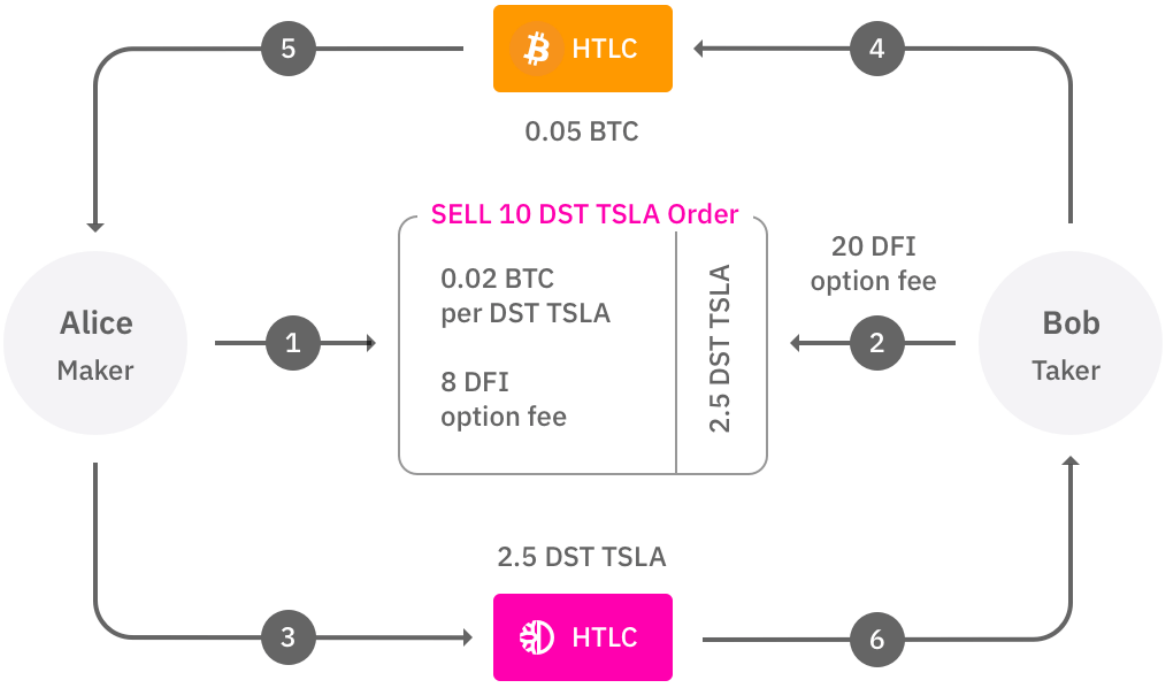
1. Buyer vs Seller
2. Maker vs Taker

Scenarios

2 scenarios:

1. DST seller maker, wants BTC
    - Paired with BTC seller taker
2. BTC seller maker, wants DST
    - Paired with DST seller taker

Scenario 1



Alice - Sells DST, wants BTC - Maker

Bob - Sells BTC, wants DST - Taker

1. Alice, as maker, puts up an order:
    - For sale: 10 TSLA DST
    - unitPrice: 0.02 *(BTC - $800 each for BTC at $40k)*
    - expiry: 2880 blocks
    - optionFeePerUnit: 8 *(DFI)*
2. Bob, as taker:

- Wants to buy just 2.5 TSLA, not all 10. *(partial order)*
- Makes an acceptance offer.
- Transmits `optionFeePerUnit * unit` 8 * 2.5 = 20 DFI alongside the order into HTLC. No payment DFI is locked up yet, only 20 DFI for optionFee
- UX: either browse orderbook, or do a market order *(easier: always match with the best price)*

3. Alice accepts the offer

   - UX side this is automatically accepted
   - Moves 2.5 TSLA from the order into DFC DST HTLC (also known as ICX Contract).
   - Properties of HTLC:
     - 2.5 TSLA locked up for 120 blocks (1 hour)
     - with H provided, where H = Hash(S) the funds for Bob can be unlocked from this HTLC.

4. Bob sees the DST HTLC, verifies the following:

   - Amount, token, time locked up and recipient is correct.
   - Waits for sufficient confirmation at DFC side to ensure finality.
   - Bob, now initiates on the Bitcoin side, HTLC for:
     - 2.5 * 0.02 = 0.05 BTC
     - With the same H. Take note that Alice knows S, but Bob does not know S.
     - Time limit is 30 minutes.
   - Bob updates ICX C on this HTLC.

5. Alice accepting Bitcoin

   - Alice confirms that HTLC properties are correct, amount, time, recipient.
   - Waits for sufficient confirmation at Bitcoin blockchain to ensure finality.
   - Claims Bitcoin by revealing S.

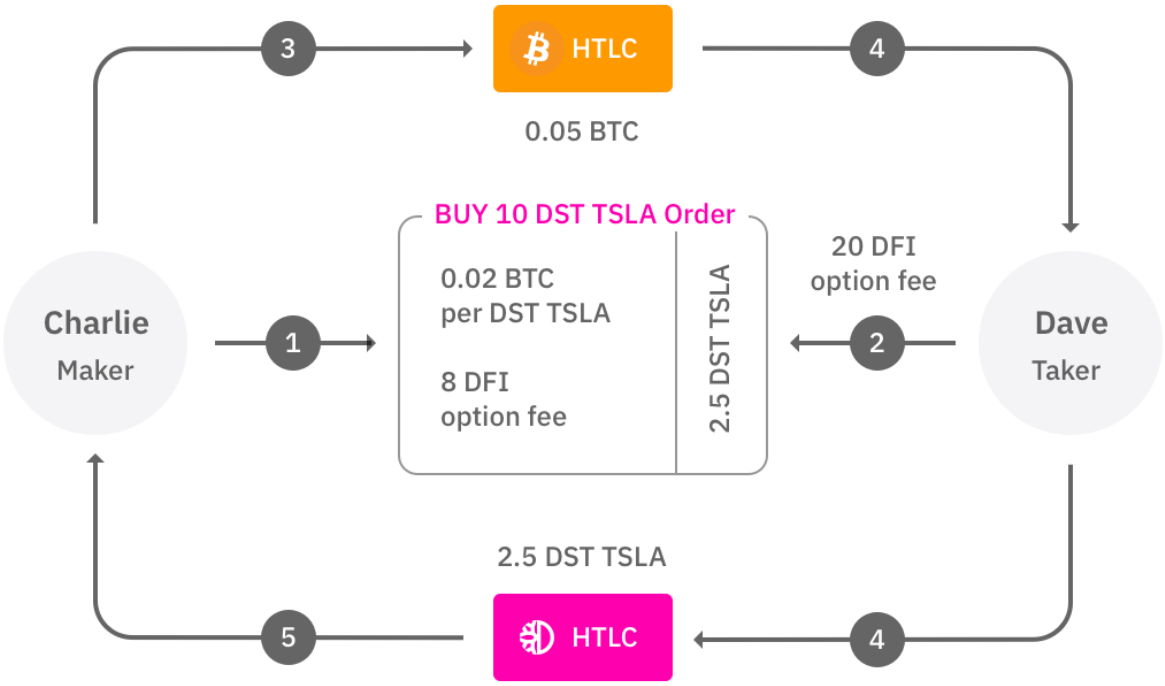
6. Bob now claims DST.

   - Bob sees that Alice has claimed and revealed S.
   - Bob claims DST too by using the same S.
   - DST is claimed on ICX C itself, this sends DST to Bob, and refunds Bob the `optionFee`.

Bad actors

1. N/A
2. N/A
3. If Alice does not accept, Bob's offer expires after 20 blocks and takes back `optionFee`.
4. If Bob does not honor the BTC side, Alice receives back her 2.5 TSLA after 1 hour + `optionFee`.

Scenario 2



Charlie - Sells BTC, wants DST - Maker

Dave - Sells DST, wants BTC - Taker

1. Charlie, as maker, puts up an order:
    - Buying: 10 TSLA DST
    - unitPrice: 0.02 *(BTC - $800 each for BTC at $40k)*
    - expiry: 2880 blocks
    - optionFeePerUnit: 8 *(DFI)*
2. Dave, as taker:
    - Wants to sell just 2.5 TSLA, not all 10. *(partial order)*
    - Makes an acceptance offer.
    - Transmits `optionFeePerUnit * unit` 8 * 2.5 = 20 DFI alongside the order into HTLC. No payment DFI is locked up yet, only 20 optionFee DFI.
    - UX: either browse orderbook, or do a market order *(easier: always match with the best price)*
3. Charlie accepts the offer
    - Charlie comes up with S. Computes H where H = Hash(S).
    - Charlie locks up BTC on the Bitcoin HTLC for:
        - 2.5 * 0.02 = 0.05 BTC
        - With H. Take note that Charlie knows S, but Dave does not know S.

        - Time limit is 1 hour (6 blocks).
    - and creates ICX Contract with H, and Bitcoin HTLC address.
    - Dave's `optionFee` is locked in ICX C.

4. Dave sees that Charlie has accepted the offer and that BTC is now locked in HTLC

    - Validates that BTC HTLS is correctly set up - amount, time, recipient
    - Waits for sufficient confirmation on BTC side to ensure finality.
    - Moves 2.5 TSLA from the order into DFC ICX C (no need to provide H as contract already know H)

5. Charlie accepts DST

    - Waits for sufficient confirmation on DFC side to ensure finality.
    - Claims DST by revealing S
    - This sends DST to Charlie and refunds `optionFee` to Dave.

6. Dave now claims BTC

    - using the same S on the BTC HTLC

Bad actors

1. N/A
2. N/A
3. If Charlie does not accept, Bob's offer expires after 10 blocks and takes back `optionFee`.
4. If Bob does not honor the BTC side, Alice receives back her 2.5 TSLA after 1 hour + `optionFee`.

Override Notice: Fees and Incentives

TODO: Cleanup this document to be coherent

The above document has parts overriden by the fees document.

Interchain Exchange fees

Game theory exploits

The following fees are suggested to prevent the following:

1. Users could potentially take up their own orders to generate fake transactions.
2. Users could potentially take up other users' orders to eliminate competition and increase own incentives returns.
3. Users could leave their orders passively and not respond to take requests.

General ideas

- Negative maker fee
- Positive taker fee
- Also the DFI made by Maker, inclusive of bonus must be less than fees paid by Taker. Without it fake transactions would be generated to game rewards.

Fees

- Previous `optionFee` is now replaced with non-refundable `takerFee`.
    - This is to deal the case where Alice refuses to accept Bob's valid HTLC *(Scenario 1)* and unfairly profits from `optionFee`.
- `takerFee` is in DFI and is fixed at consensus-set (`takerFeePerBTC` * BTC * `DEX DFI per BTC rate`).
- `makerDeposit` being the same as `takerFee`.
    - It is refundable upon successful swap.
- `makerIncentive`
    - `25% * takerFee`
- `makerBonus`
    - `50% * takerFee` if it's BTC parity trade. BTC DSTs eligible for `makerBonus` are consensus-set.

DST maker exiting to Bitcoin

1. Maker, pays no fee, but locks up DST *(consensus-verifiable)*.

2. Taker makes offer, locks up `takerFee`.
    - If offer is not accepted after timeout, `takerFee` is refunded.
3. Maker accepts offer, locks up DST *(consensus-verifiable)*, burns `makerDeposit` & `takerFee`.
    - If partial amount from offer is accepted, refunds unused `takerFee` proportionally back to Taker.
4. Taker locks up BTC *(non-consensus-verifiable)* and updates ICXC on BTC HTLC.
5. Maker verifies that BTC HTLC is correct, claims BTC.
    - If Taker locks up incorrectly, both parties lose DFI *(neither party wins)*
    - If Maker does not verify, both parties lose DFI *(neither party wins)*
6. Taker claims DST on ICXC.
    - Mints `makerDeposit` *(refund)*, `makerIncentive` & `makerBonus` *(if any)* to Maker.
    - Validate that the amounts are correct – no additional tokens are generated.

BTC maker entering from Bitcoin

1. Maker, pays no fee.
2. Taker makes offer, locks up `takerFee`.
    - If offer is not accepted after timeout, `takerFee` is refunded.
3. Maker accepts offer, locks up `makerDeposit`.
4. Taker locks up DST *(consensus-verifiable)*, burns both `takerFee` and `makerDeposit`.
    - If Taker does not do it, burns `takerFee` and refund `makerDeposit` to Maker.
5. Maker locks up BTC *(non-consensus-verifiable)* and updates ICXC on BTC HTLC.
6. Taker verifies that BTC HTLC is correct, claims BTC.
    - If Maker locks up incorrectly, both parties lose DFI *(neither party wins)*
    - If Taker does not verify, both parties lose DFI *(neither party wins)*
7. Maker claims DST on ICXC
    - Mints `makerDeposit` *(refund)*, `makerIncentive` & `makerBonus` *(if any)* to Maker.
    - Validate that the amounts are correct – no additional tokens are generated.

Loan and Decentralized Tokenization

Loan is be offered by operator to allow decentralized price-tracking tokens to be created by users.

Before Operator model is ready, it uses only 1 default opspace and requires foundation authorization, that acts on behalf of community.

Possibility of tokens being offered in a decentralized manner can be of many types, including but not limited to:

1. Cryptocurrency such as `BTC`, `ETH`, `LTC`, etc.
2. Stablecoin such as `USD`, `EUR` tokens, etc.
3. Stocks token such as `TSLA`, `MSFT`, etc.
4. Other asset tokens.

Concepts

Collateralization ratio

Loan on DeFiChain operates on over-collateralization mechanism. User first has to open a vault to mint tokens.

Collateralization is calculated for each vaults, using the following formula:

```
Collateralization ratio =
    Total effective value of collateral /
    (Total value of tokens minted + Total interest)
```

For example, if a vault holds $200 worth of collateral asset, with $90 minted tokens and $10 of total interest accrued, the collateralization ratio is 200 / (90 + 10) = 2.

This is not to be confused with *collateralization factor*.

Collateralization factor

Assets that are accepted for collateralization could be accepted at different collateralization factor, from 0% to 100%. When an asset collateralization factor is 100%, the entirety of the asset's value contributes to the collateralization value of the vault, however, if an asset collateralization factor is say 50%, then only half of its value contributes to the total vault's collateral.

For example if `DOGE` is accepted at 70% collateralization factor, $100 of `DOGE` would contribute to $70 of collateral value in a vault.

This is not to be confused with *collateralization ratio*.

Operator

Operator plays to role to decide on the following:

1. Collateral tokens and their respective collateralization values.
    - For instance, an Operator can decide to allow users to start a Vault in their opspace with the following collateralization factors:
        - `DFI` at 100% collateralization factor
        - `BTC` at 100% collateralization factor
        - `DOGE` at 70% collateralization factor

Operator, also via opspace's governance values, can decide the following:

- `LOAN_LIQUIDATION_PENALTY`: 0.05 (default), for 5% liquidation penalty. This is converted into DFI upon liquidation and burned.

Loan scheme

Loan scheme allows an operator to set up different loan schemes under the same Operator. Loan schemes allows an operator to define the following:

- Collateral token and different collateralization factor
- Loan interest rates

For example, a series of loan schemes could be made available:

1. Min collateralization ratio: 150%. Interest rate: 5% APR.
2. Min collateralization ratio: 175%. Interest rate: 3% APR.
3. Min collateralization ratio: 200%. Interest rate: 2% APR.
4. Min collateralization ratio: 350%. Interest rate: 1.5% APR.
5. Min collateralization ratio: 500%. Interest rate: 1% APR.
6. Min collateralization ratio: 1000%. Interest rate: 0.5% APR.

Vault owner can define which loan scheme to subscribe to during initial vault creation and can move it freely after that. Take caution though that if you move to a scheme with lower collateralization ratio, liquidation might be triggered.

Interest rate

Interest rate for loan consists of 2 parts:

- Vault interest, based on loan scheme of individual vaults
- Token interest, based on loan tokens, e.g. `TSLA` token might have its own interest that is chargeable only for `TSLA` token.

DeFi fees are burned. Fees are typically collected in the form of the loan token repayment, and automatically swapped on DEX for DFI to be burned.

For example, if a loan of 100 `TSLA` is taken out and repaid back exactly 6 months later with the APR of 2%, the user will have to repay 101 `TSLA` during redemption to regain full access of collateral in the vault.

100 `TSLA` will be burned as part of the repayment process. 1 `TSLA` that forms the interest payment will be swapped on DEX for DFI. The resulting DFI will burned. All of these occur atomically as part of DeFiChain consensus.

Vault

User is able to freely open a vault and deposit tokens to a vault. Vault is transferable to other owners, including being controlled by multisig address.

Collateral DFI requirement

Vault's collateral requires at least 50% of it to be DFI.

When depositing non-DFI collateral to a vault, it needs to ensure that the resulting DFI proportion of for vault's collateral is at least 50% or more, or the deposit transaction should fail.

This requirement is only checked upon deposited and does not play a role in liquidation. For instance, if at the time of posit, a vault's DFI collateral is 52%, but falls to 49% without any further deposit. This WILL NOT trigger liquidation as long as vault's minimum collateralization ratio condition is still met.

States

- "active" : Vault is above its collateralization ratio and both collaterals and loans price feeds are valid.
- "frozen" : Any of collateral or loan price feed is invalid. No action can be undertaken while vault is frozen.
- "inLiquidation" : Vault is under liquidation and has undergoing auctions.

- "mayLiquidate" : Vault will liquidate at the next price update.

Liquidation

Liquidation occurs when the collateralization ratio of a vault falls below its minimum. Liquidation is crucial in ensuring that all loan tokens are sufficiently over-collateralized to support its price.

For Turing-complete VM-based blockchain, liquidation requires the assistance of publicly run bots to trigger it, in exchange for some incentives.

There are two problems with that:

- During major price movements, the network fee, or commonly known as gas price, would sky rocket due to a race by both the vault owners frantically trying to save their vaults from getting liquidated, and by the public liquidators trying to trigger liquidation.
- If liquidations are not triggered on time, tokens that are generated as a result of loan runs the risk of losing its values. Worse scenario would see it getting into a negative feedback loop situation creating a rapid crash of token values.

DeFiChain, a blockchain that's built for specifically for DeFi, enjoy the benefit of automation and can have liquidation trigger automatically on-chain. This address the above two problems.

Methodology

On every block, the node would evaluate all vaults and trigger the following automatically for liquidation.

When collateralization ratio of a vault falls below minimum collateralization ratio, liquidation would be triggered. During liquidation collateral auction is initiated.

When a vault is in liquidation mode, it can no longer be used until all the loan amount is repaid before it can be reopened. Interest accrual is also stopped during liquidation process.

Collateral auction

The entirety of loan and collateral of a vault if put up for auction. As the first version of collateral auction requires bidding of the full amount, auction is automatically split into batches of *around* $10,000 worth each to facilitate for easier bidding.

Example

A vault, that requires a minimum of 150% collateralization ratio contains $15,000 worth of collateral, which consists of $10,000 worth of DFI and $5,000 worth of BTC, and the total loan, inclusive of interest, is $11,000 worth.

- Collateral: $10,000 DFI + $5,000 BTC
- Loan: $10,000 `TSLA` + $1,000 interest (`TSLA`)

Collateralization ratio = 15k / 11k = 136.36% less than the required 150%. Liquidation is triggered automatically by consensus. Auction will be initiated with the intention to liquidate $15k of collateral to recover $11k of loan.

Liquidation penalty is defined as `LOAN_LIQUIDATION_PENALTY`. The amount to be recovered should also include the penalty. If `LOAN_LIQUIDATION_PENALTY` is 0.05, the total amount to be recovered is thus $11k * 1.05 = $11,550

Total collateral worth is $15k, it will thus be split into 2 batches, each not exceeding $10k:

1. 2/3 of the whole vault, worth $10k of collateral.
2. 1/3 of the whole vault, worth $5k of collateral.

Specifically the following:

1. ($6,667 DFI and $3,333 BTC) for $7,700 `TSLA`
2. ($3,333 DFI and $1,667 BTC) for $3,850 `TSLA`

`TSLA` recovered at the conclusion of the auction will be paid back to the vault. Once all loan is repaid, vault will exit liquidation state, allowing its owner to continue to use it. It will, however, not terminate all opened auctions. Opened auctions see through to their conclusion.

Collateral auction

Collateral auction will run for 720 blocks (approximately 6 hours) with the starting price based on liquidation vault's ratio.

Using the same illustration, Auction 1: ($6,667 DFI and $3,333 BTC) for $7,700 `TSLA`

Anyone can participate in the auction, including the vault owner by bidding for the entirety of the auction with higher amount. To prevent unnecessary auction sniping at closing blocks, each higher bid, except the first one, has to be at least 1% higher than previous bid.

A user could bid for the same auction by offering $7,800 worth of `TSLA` tokens. During bidding, the bid is locked and refunded immediately when outbid. Top bid is not cancelable.

At the conclusion of the auction, the entirety of the for sale collateral will be transferred to the winner – in this case the winner would be paying $7,800 for $10,000 worth of BTC and DFI.

$7,800 worth of `TSLA` token will be processed as follow:

- Paid back to vault = $7,800 worth of `TSLA` / (1 + `LOAN_LIQUIDATION_PENALTY`) = 7800 / 1.05 = $7,428.57 `TSLA` token
- The remaining, i.e. liquidation penalty, will be swapped to DFI and burned. $371.43 worth of `TSLA` be swapped as follows and trackably burned:
    1. `TSLA` to `USD` from `TSLA-USD` DEX.
    2. `USD` to `DFI` from `USD-DFI` DEX.

If auctions yield more `TSLA` than the vault's loan, it will be deposited back to the vault as part of vault's asset, but not collateral. It can be withdrawn from the vault after vault exits liquidation state by vault's owner and carries no interest.

Multiple loan tokens

If a vault consists of multiple loan tokens, they should be auctioned separately.

For instance, a liquidating vault that consists of the following:

- Collateral: 1 BTC & 100 DFI
- Loan: $10k worth of USD * $40k worth TSLA

would be liquidated as follows:

- (0.2 BTC & 20 DFI) for $10k worth of USD
- (0.8 BTC & 80 DFI) for $40k worth of TSLA

and further split into $10k batches.

Failed auction

Auction that expires after 720 blocks without bids will be reopened immediately based on the remaining amount of collateral and loan that the vault receives due to conclusion of other auction batches.

RPC

Operator

Requires Operator authorization. Before Operator model is ready, it uses only 1 default opspace and requires foundation authorization.

Loan scheme

1. `createloanscheme DATA`
    - `DATA` (JSON) can consist of the following:
        - `mincolratio`: e.g. 175 for 175% minimum collateralization ratio. Cannot be less than 100.
        - `interestrate`: Annual rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be > 0.
        - `id`: Non-colliding scheme ID that's unique within opspace, e.g. `MIN_175`
2. `updateloanscheme MIN_COL_RATIO INTEREST_RATE SCHEME_ID [ACTIVATE_AFTER_BLOCK]`
    - Update loan scheme details.
    - `ACTIVATE_AFTER_BLOCK` *(optional)*: If set, this will only be activated after the set block. The purpose is to allow good operators to provide sufficient warning.
3. `destroyloanscheme SCHEME_ID [ACTIVATE_AFTER_BLOCK]`
    - Destroy a loan scheme.
    - `ACTIVATE_AFTER_BLOCK` *(optional)*: If set, this will only be activated after the set block. The purpose is to allow good operators to provide sufficient warning.
    - Any loans under the scheme will be moved to default loan scheme after destruction.
4. `setdefaultloanscheme SCHEME_ID`
    - Designate a default loan scheme from the available loan schemes.

General

1. `setcollateraltoken DATA`
    - `DATA` (JSON)
        - `token`: Token must not be the same decentralized token is issued by the same operator's loan program.

        - `factor`: A number between `0` to `1`, inclusive. `1` being 100% collateralization factor.
        - `fixedIntervalPriceId`: Token/currency pair to track token price.
        - `activateAfterBlock` *(optional)*: If set, this will only be activated after the set block. The purpose is to allow good operators to provide sufficient warning to their users should certain collateralization factors need to be updated.
        - This very same transaction type is also used for updating and removing of collateral token, by setting the factor to `0`.

2. `setloantoken DATA`

    - Creates or updates loan token.
    - `DATA` (JSON)
        - `symbol`Token's symbol (unique), not longer than 8
        - `name` Token's name (optional), not longer than 128
        - `fixedIntervalPriceId`: Token/currency pair to track token price.
        - `mintable` (bool): When this is `true`, vault owner can mint this token. (defaults to true)
        - `interest`: Annual rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be >= 0. Default: 0.
    - To also implement `updateloantoken` and `listloantokens`.

Public

Requires no authorizations.

1. `getloaninfo`

    - Returns the attributes of loans offered by Operator.
    - Includes especially the following:
        - Collateral tokens: List of collateral tokens
        - Loan tokens: List of loan tokens
        - Loan schemes: List of loan schemes
        - Collateral value (USD): Total collateral in vaults
        - Loan value (USD): Total loan token in vaults

2. `listloanschemes`

    - Returns the list of attributes to all the available loan schemes.

Vault

Vault-related, but does not require owner's authentication.

1. `createvault [OWNER_ADDRESS] [SCHEME_ID]`

- Create a vault for `OWNER_ADDRESS`.
- No `OWNER_ADDRESS` authorization required so that it can be used for yet-to-be-revealed script hash address.
- 2 DFIs needed for createvault.
- 1 DFI burned, and 1 added to collateral.
- The last 1 added to collateral will be reclaimed on closevault.

2. `deposittovault VAULT_ID TOKEN_TO_DEPOSIT`

   - Deposit accepted collateral tokens to vault.
   - Does not require authentication, anyone can top up anyone's vault.
   - `TOKEN_TO_DEPOSIT` should be in similar format as other tokens on DeFiChain, e.g. `23.42@USDC` and `0.42@DFI`.
   - Also take note on the >= 50% DFI requirement, when depositing a non-DFI token, it should reject if the total value of the vault's collateral after the deposit brings DFI value to less than 50% of vault's collateral.
   - This also means that the first deposit to a vault has to always be DFI.

3. `paybackloan DATA`

   - `DATA` (JSON)
     - `vaultId`: loan's vault ID.
     - `from`: Address containing repayment tokens.
     - `amounts`: Amounts to pay back in amount@token format.
   - Pay back of loan token.
   - Only works when there are loans in the vault.
   - Refunds additional loan token back to original caller, if 51.1 `TSLA` is owed and user pays 52 `TSLA`, 0.9 `TSLA` is returned as change to transaction originator (not vault owner).
   - Does not require authentication, anyone can payback anyone's loan.

4. `getvault [VAULT_ID]`

   - Returns the attributes of a vault.
   - Includes especially the following:
     - Owner address
     - Loan scheme ID
     - `state` Can be one of `active`, `frozen`, `inLiquidation` and `mayLiquidate`. See state

5. `listvaults`

   - Returns the list of attributes to all the created vaults.

Vault owner

Requires ownerAddress authentication, and vault MUST NOT be in liquidation state.

1. `updatevault VAULT_ID DATA`
    - `DATA` (JSON) may consists of the following:
        - `scheme`: Allows vault owner to switch to a different scheme
        - `ownerAddress`: Transfers vault's ownership to a new address.
2. `withdrawfromvault VAULT_ID TO_ADDRESS AMOUNT`
    - Withdraw collateral tokens from vault.
    - Vault collateralization ratio must not be less than `mincolratio` of vault's scheme.
    - Also used to withdraw assets that are left in a vault due to higher auction yield. See Collateral auction
3. `takeloan DATA`
    - `DATA` (JSON)
        - `vaultId`: ID of vault used for loan.
        - `to`: Address to transfer tokens (optional).
        - `amounts`: Amount in amount@token format.
    - Vault collateralization ratio must not be less than `mincolratio` of vault's scheme.

Auction

Requires no additional authentications.

1. `listauctions`
2. `auctionbid VAULT_ID INDEX FROM AMOUNT`
    - `VAULT_ID`: Vault id.
    - `INDEX`: Auction index.
    - `FROM`: Address to get tokens.
    - `AMOUNT`: Amount of amount@symbol format.
    - `TOTAL_PRICE` of token in auction will be locked up until the conclusion of the auction, or refunded when outbid.

Operator

Generalization

Generalization effort in DeFiChain aims at separation between the role of blockchain and operator, the party in charge of offering financial products.

The generalization goals are to allow anyone to create and run as operator on DeFiChain, without requiring any prior permissions from DeFiChain Foundation or its developers.

Opspace

Operator operates exclusively within their Opspace and does not interfere with the operations of other Opspaces and entire blockchain ecosystem.

To ensure operation continuity, the current asset tokens, and any other offerings that are available prior to the introduction of Operator, will be migrated to the first Opspace when the functionality is available.

Role

The role and functionalities available to operators include:

1. Price oracle

- Setting of price feeds.
- Appointing of oracles.

2. Asset token

- Determining and setting of asset tokens that are available within an opspace.
- Previously referred to as "DeFi Asset Token" on white paper.

3. Decentralized loan

- Determining loan attributes: collateral types, interest rates, etc

4. Futures

- Determining future offerings & attributes

... and others as they are made available.

Operator is also free to determine their own Operator fee for all the products that they provide.

Note that operators may only want to participate in a subset of the operations available, for example an operator may only be interested in offering decentralized loan but not interested in offering futures.

Operations

RPC is designed work with multisignatory opcodes of Bitcoin's script, allowing operator to have clear governance over its policies.

1. `createoperator`
    - Costs 1000 DFI in DeFi fee, burned.
2. `updateoperator`

Price oracle

Even though Oracle could technically provide any off-chain data, the initial version of Oracle focuses on numeric data especially price feeds to augment various features and functionalities on DeFiChain, especially future, option & loan.

Economics & game theory

To provide a fully robust trustless oracle service requires components such as reputation, penalty, and various other mechanics, it is not DeFiChain's current focus to build a full oracle system, at least at this stage. DeFiChain utilizes Operator and free economy to ensure that oracles are run as fairly as possible. It is Operator's interest to ensure that oracles that they appoint are fairly run, and users are free to select operators that they like.

Mechanics

While truthfullness of the data cannot be assertained on the blockchain alone, the goals of the design should focus on building a *robust* oracle system that should continue to function fine with a few minotirty oracle outages.

Price feeds

Price feed should be 8 decimal places.

Requires foundation authorization.

1. `createpricefeed BTC_USD_PRICE`
2. `removepricefeed BTC_USD_PRICE`

Oracle appointments

Requires Operator authorization. Before Operator model is ready, it uses only 1 default opspace and requires foundation authorization.

1. `appointoracle ADDRESS '["BTC_USD_PRICE", "ETH_USD_PRICE"]' WEIGHTAGE`
    - returns `oracleid`. `oracleid` should be a deterministic hash and not incremental count.
    - `WEIGHTAGE` is any number between 0 to 100. Default to 20.

2. `removeoracle ORACLEID`
3. `updateoracle ORACLEID ADDRESS '["BTC_USD_PRICE", "ETH_USD_PRICE"]' WEIGHTAGE`

Price feeds operations

Requires appointed Oracle authorization.

1. `setrawprice TIMESTAMP '{"BTC_USD_PRICE": 38293.12}'`
    - Raw price update should only be valid if timestamp is within one hour from the latest blocktime.
    - timestamp is there so that updates that are stuck in mempool for awhile does not get accepted after timeout.

General usage

General usage, requires no authorizations.

1. `listlatestrawprices BTC_USD_PRICE`

    - Also supports `listlatestrawprices` to list across all feeds.
    - Returns an array of all the latest price updates, e.g.

```
[
    {
        "pricefeed": "BTC_USD_PRICE",
        "oracleid": "ORACLEID",
        "weightage": 50,
        "timestamp": "timestamp of the last update",
        "rawprice": 38293.12000000,
        "state": "live"
    }
]
```

    - Price updates that are older than 1 hour from the latest blocktime would have its state to be `expired`.
    - It is also imperative that the node is not storing all raw price data on database, it only needs to know the latest raw price for each oracles.

2. `getprice BTC_USD_PRICE`

    - Returns number with aggregated price.
    - Aggregate price = Sum of (latest rawprice of live oracles * weightage) / Sum (live oracles' weightage)

- If no live oracle is found for the price, it should throw an error. It MUST NOT return 0 or large number.

3. `listprices`
    - Same as above, but an array of all feeds.

DFIP-2203

Proposal

For the community discussion that led to the addition of this feature, please have a look at the Reddit post and hear the Twitter Space.

For the proposal, refer to this link

DFIP summary :

- Introduce future trading for settling dToken prices once a week (7*2880 blocks).
- Establish 2 different Future contracts - one for the premium case and on for the discount case.
- Range for settlement should be +/-5%, which allows anticipating the next price in case of closed markets (constant oracle price)
- Premium future should trade 5% above the oracle price - you can buy dTokens with a 5% fee)
- Discount future should trade 5% below the oracle price - you can sell dTokens with a 5% discount)
- Futures will be a weak binding of DEX price to oracle price: between settlement time higher/lower deviations are possible
- Introducing an additional swap fee of 0.1% for every dToken pool to burn dTokens. The fee should be conducted for both swap directions.
- Strengthen dUSD by counting them in the same way as the mandatory 50% DFI in vaults with fix price of $0.99.

Related PRs:

- DEX fee PR
- DUSD as collateral PR
- Future swap contract PR

Governances variables

v0/params/dfip2203/active

This variable determines wether the DFIP is activated or not. When disabled, no more deposit to the smart contract address will be accepted nor any swap will be executed.

v0/params/dfip2203/reward_pct

The percent of premium deducted at the moment of the swap.

v0/params/dfip2203/period_blocks

The block interval between two future swaps.

v0/token/<token_id>/dfip2203_disabled

Defaults to false. Used to blacklist certain token.

RPC commands

getfutureswapblock

Get the next block when the contract will be executed.

```
defi-cli getfutureswapblock
895100
```

futureswap

Parameters :

- `address` : Address to fund contract and receive resulting token.
- `amount` : Amount to send in amount@token format.
- `destination` : Optional. Expected dToken if DUSD amount supplied.

Deposit DUSD or dTokens to be swapped at the next block interval.

```
defi-cli futureswap <address> <amount> <destination>
fd55837215c7ff29ba43afb196ffbfba8c20f16eb9dbba30ba829853f15b2aeb
```

withdrawfutureswap

Parameters :

- `address` : Address to fund contract and receive resulting token.
- `amount` : Amount to send in amount@token format.
- `destination` : Optional. Expected dToken if DUSD amount supplied.

Withdraw DUSD or dTokens previously deposited.

```
defi-cli withdrawfutureswap <address> <amount> <destination>
bceb216add60c38888d102c737d73d3a559fc214bbe18cab92848a8d31586944
```

listpendingfutureswaps

List all pending futures.

```
defi-cli listpendingfutureswaps
[{'owner': 'mg1qAZBb28tuWpxNmbRvZha9zhUpEQ5uVx', 'source': '1.00000000@MSFT',
'destination': 'DUSD'}, {'owner': 'mhe4v9VMpinWfJpgUqpmyEXeUXc1AdFoVC',
'source': '1.00000000@GOOGL', 'destination': 'DUSD'}, {'owner':
'mk5RKGj4nNiVPhKvrVqkUh7oX2oGXvtXc5', 'source': '1.00000000@TSLA',
'destination': 'DUSD'}, {'owner': 'msyMCXeC12XwzJ8CbZTbDyDAem3PsWFrRJ',
'source': '1.00000000@TWTR', 'destination': 'DUSD'}]
```

getpendingfutureswaps

Get all pending futures for a specific address.

Parameters :

- `address` : Address to get futures prices for.

```
defi-cli getpendingfutureswaps <address>
{'owner': '<address>', 'values': [{'source': '1.00000000@MSFT', 'destination':
'DUSD'}]}
```

listgovs

All the governance variables related to this DFIP can be accessed via `listgovs`

```
defi-cli listgovs
{
    ...,
    "ATTRIBUTES": {
    "v0/params/dfip2203/active": "true",
    "v0/params/dfip2203/reward_pct": "0.05",
    "v0/params/dfip2203/block_period": "20160",
    "v0/token/10/dfip2203_disabled":"true",
    }
}
```

Guides

Blockchain Node - defid

DeFiChain is a distributed network of computers running software (known as nodes). DeFiChain nodes are able to

- Store all blockchain data
- Serve data to lightweight clients
- Validate and propogate blocks and transactions

Every DeFiChain node runs an implementation of the DeFiChain protocol, and can be run on local and cloud machines. The recommended requirements for running the reference `defid` node are

- Desktop or laptop hardware running recent versions of Windows, Mac OS X, or Linux.
- Atleast 30 GB of disk space (Minimum: ~8GB)
- Atleast 8 gigabytes of physical memory (RAM) (Minimum: ~1GB, but isn't recommended and sufficient swap is required)
- 4 GB of swap space (Minimum: None, but highly recommended to give room for memory fragmentation spikes)

There are two ways start using the node - using the compiled releases or compiling the souce code on your own.

Using compiled releases

Download the latest available release from Github. Extract the archive by using an archive manager, or running the following command in a terminal

```
tar xzf <filename>.tar.gz # replace <filename> with the filename downloaded file
```

You can run the following install command to globally install the DeFiChain binaries.

```
sudo install -m 0755 -o root -g root -t /usr/local/bin <defichain-path>/bin/* #
replace <defichain-path> with the path to the root of the extracted folder
```

Proceed to Initial Block Download.

Initial Block Download

Once you have the required binaries available, you can start the node by running the following commands in the folder with the executables.

Linux and macOS: `./defid` Windows: `defid.exe`

This will start the node and the software will begin syncing blockchain data. Your wallets and transaction history will not be visible until the blockchain has been completely synced and up to date with the rest of the nodes. Currently the blockchain takes up **around 4GB** of disk space.

In order to speed up the process, you can download a snapshot from AWS.

1. Stop the `defid` process.
2. Go to an AWS bucket and copy the filename of the latest (bottom most) snapshot.

- America
- Asia
- Australia
- Europe

3. Replace `index.txt` in the URL with the filename from the previous step.
4. Extract the contents of the downloaded `.zip` file.
5. Copy the extracted contents into the `.defi` folder.
    - Windows: `C:\Users\<username>\AppData\Roaming\DeFi Blockchain`
    - Linux: `~/.defi/`
    - macOS: `/Users/<username>/Library/Application\ Support/DeFi`
6. Now restart the `defi` process. There will be a short syncing process to get the node up to date with the rest of the network.

Interacting with the node

Once `defid` is running, you interact with the node on a different terminal using the `defi-cli` application. You can run `./defi-cli help` to see a list of available commands.

Proceed to Using a wallet to learn how to get started with using the node.

Upgrading the node

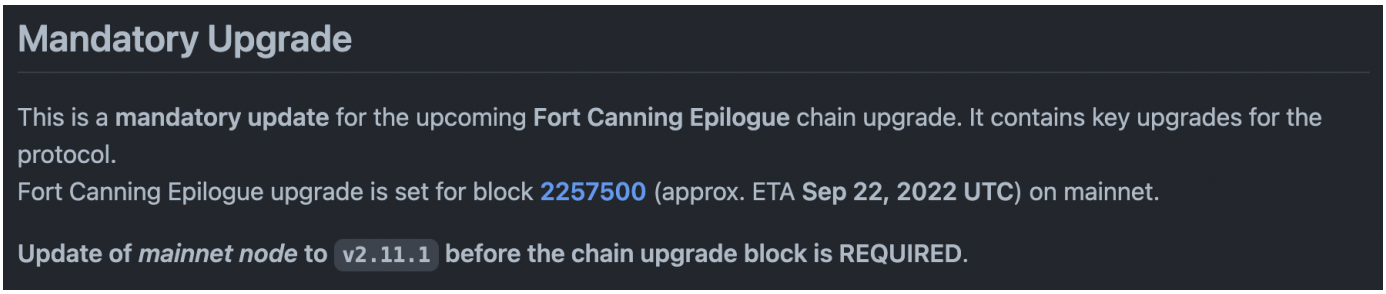
Masternode operators and other users who want to upgrade their nodes to the latest release to access the latest features and guarantee compatibility with the latest version of the DeFi Blockchain.

> **NOTE**
>
> DeFiChain Wallet users do not need to manually update the node as the wallet will handle the update for you.

There are two types of releases:

- **Mandatory**: contain breaking changes. You must upgrade before the upgrade block height for your node to remain in sync.
- **Optional**: optional but *highly reccomended*, usually include performance improvements.

Mandatory upgrades are marked in the release notes on Github along with the block height before which you should upgrade.



Mandatory upgrade note

`defid` updates are usually drop in. Users can simply download and extract the latest `.zip` files from Github and run them instead of the older binaries.

If you upgrade your node, have a wallet, and use a snapshot to reduce sync time, you should run a `-rescan` (and `-spv_resync` if you hold BTC) when running for the first time to update wallet balances.

Some releases might require additional parameters to be run on the first time with the new version, these will be stated in the release notes.

- Reindex required: run with `-reindex` flag
- Rescan required: run with `-rescan` flag
- Fresh sync: delete all files in and directories in the `.defi` folder except `wallets` and relaunch

Compiling from source

Using `make.sh` (Debian/Ubuntu only)

By running `./make.sh build` will download install required dependencies and run the complete compilation process. Run `make install` at the end of the process to make the compiled binaries available globally.

Compiling manually

1. Install `gcc` and `boost` and other dependecies from your distribution's package manager.

   Arch Linux

   ```
   sudo pacman -S gcc boost libevent python
   ```

   Debian/Ubuntu

   ```
   sudo apt-get install build-essential libtool autotools-dev automake pkg-config bsdmainutils python3 curl ibssl-dev libevent-dev libboost-system-dev libboost-filesystem-dev libboost-chrono-dev libboost-test-dev libboost-thread-dev
   ```

2. Clone the latest version of `ain` from GitHub.

   ```
   git clone https://github.com/DeFiCh/ain.git
   cd ain
   ```

3. Install Berekely DB.

   ```
   ./contrib/install_db4.sh `pwd`
   ```

   Once installation is complete, take note of the commands provided at the end of the build log.

```
db4 build complete.

When compiling defid, run `./configure` in the following way:

export BDB_PREFIX='/home/DeFiCha/ain/contrib/db4'
./configure BDB_LIBS="-L${BDB_PREFIX}/lib -ldb_cxx-4.8" BDB_CFLAGS="-
I${BDB_PREFIX}/include" ...
```

4. Now, run the export command from the DB compilation, then run `autogen.sh` and finally run the configure command from above.

```
export BDB_PREFIX='/home/DeFiCha/ain/contrib/db4'
./autogen.sh
./configure BDB_LIBS="-L${BDB_PREFIX}/lib -ldb_cxx-4.8" BDB_CFLAGS="-
I${BDB_PREFIX}/include"
```

5. We can now compile the node. Run `make` in the `ain` root directory.

```
make #
make -j "$(($(nproc)))" # for multicore CPUs
```

`defid` and other DeFiChain executables will now be in the `src/` directory. You can run `make install` to access the compiled binaries globally.

Running tests

If you make any changes to the DeFiChain code, you should always update and run tests in order to verify functionality and prevent regressions.

defid tests are located in the following locations

- `/src/test`: defid unit tests
- `/src/wallet/test`: wallet unit tests
- `/test/functional`: test using RPC and P2P interfaces
- `/test/util`: test the defi utilities
- `/test/lint`: static analysis checks

Running tests in requires defid to be compiled locally. Click here to learn more about how to compile locally.

Unit tests (including wallet unit tests)

Unit tests are compiled unless disabled in `./configure`. After configuring and compiling, tests can be run by running `make check`, or by running the `test_defi` executable in `src/test`.

```
make check              # run tests using make
./src/test/test_defi    # run tests using test_defi from project root
```

To run an unit test suite, you can pass the filename (without the path and extension) to the `--run_test` argument in `test_defi`.

```
./src/test/test_defi --run_test=blockchain_tests
```

You can also run an individual unit test (denoted by `BOOST_AUTO_TEST_CASE` in test `.cpp` files) by specifying the test name after a `/`. For example, running the `get_difficulty_for_very_low_target` in `blockchain_tests.cpp` with `all` log level.

```
./src/test/test_defi --
run_test=blockchain_tests/get_difficulty_for_very_low_target --log_level=all
```

`test_defi` has multiple configuration options, which can be listed by running `test_defi --help`.

Functional tests

The ZMQ functional test requires a python ZMQ library. Install the `pyzmq` dependency through pip or your system's package manager.

```
pip install pyzmq
```

Once you have compiled defid, you can run tests by

- directly running the test's `.py` file

  e.g. to run the `feature_block` tests, we would run

  ```
  python test/functional/feature_block.py
  ```

- using `test_runner.py`

  e.g. to run the `feature_block` tests, we would run

  ```
  python test/functional/test_runner.py feature_block.py
  ```

  Multiple tests can be run using the test runner.

  ```
  python test/functional/test_runner.py <testname1> <testname2> <testname3>
  ...
  ```

  Wildcard test names can be passed, if the paths are coherent and the test runner is called from a `bash` shell or similar that supports globbing. For example, to run all the wallet tests

  ```
  python test/functional/test_runner.py test/functional/wallet*     # called
  from project root
  python functional/test_runner.py functional/wallet*               # called
  from the test/ directory
  python test_runner.py wallet*                                     # called
  from the test/functional/ directory
  ```

  The paths must be consistent. The following does not work as `wallet*` is not defined at the project root.

  ```
  test/functional/test_runner.py wallet*
  ```

  Combinations of wildcards can be passed. For example,

```
test/functional/test_runner.py ./test/functional/tool*
test/functional/mempool*
```

```
test_runner.py tool* mempool*
```

Run the regression test suite with

```
test/functional/test_runner.py
```

Run all possible tests with

```
test/functional/test_runner.py --extended
```

By default, up to 4 tests will be run in parallel by test_runner. To specify how many jobs to run, append `--jobs=n`

The individual tests and the test_runner harness have many command-line options.

- `--ansi`: use ANSI colors and dots in output (enabled by default when standard output is a TTY)
- `--combinedlogslen`: on failure, print a log (of length n lines) to the console, combined from the test framework and all test nodes
- `--coverage`: generate a basic coverage report
- `--ci`: run checks and code that are usually only enabled in a continuous integration environment
- `--exclude`: specify a comma separated list of tests to exclude
- `--extended`: run the extended test suite in addition to the basic tests
- `--help`: print help text
- `--jobs`: how many test scripts to run in parallel, defaults to 4
- `--keepcache`: retain the cache from the previous testrun
- `--quiet`: only prints dots, results summary and failure logs
- `--tmpdirprefix`: directory for datadirs
- `--failfast`: stop execution after the first test failure
- `--filter`: run scripts by filtering using regex

Util tests

Util tests can be run locally by running `test/util/defi-util-test.py`. Use the `-v` option for verbose output.

Lint tests

Lint tests requires codespell and flake8 dependencies, you can install them using pip or your system's package manager.

```
pip install codespell flake8
```

Individual tests can be run by directly calling the test script.

```
./test/lint/<filename>.sh
```

You can run all the shell-based lint tests by running `lint-all.sh`.

```
./test/lint/lint-all.sh
```

Using a wallet

This document will go over how to create a wallet and addresses in order to send and recieve DFI and dTokens.

Using testnet

In this tutorial we will use the testnet network in order to prevent any possibility of financial loss. Testnet is a separate network from mainnet for the purpose of testing. To use the testnet, stop the `defid` process, if any, and re-run it with the `-testnet` flag.

Linux and macOS: `./defid -testnet`

Windows: `defid.exe -testnet`

To verify that you are on the testnet, run `./defi-cli -testnet getblockchaininfo`. You should see an output similar to the following

```
{
  "chain": "test",
  "blocks": 919519,
  "headers": 934011,
....
  },
  "warnings": ""
}
```

If you see `"chain": "test"`, you are on the testnet!

Creating an address

To create an address, run

```
./defi-cli -testnet getnewaddress
```

This will return a new DeFiChain address that you can use to recieve DFI and dTokens.

```
tnLRVU32vCfGD6...
```

You will also require the private keys to this address in order to access your funds. Use the `dumpprivkey` command to retrieve the private key.

```
./defi-cli -testnet dumpprivkey
```

This will return the corresponding private key to the input address.

```
cRomqYvttzcXHq...
```

Do not share your private keys with anyone.

You can run the `listreceivedbyaddress` command to see available addresses and the amount of DFI available to each address.

```
./defi-cli -testnet listreceivedbyaddress 1 true
```

Here the first parameter sets the minimum number of confirmations required to include the transaction, and the second parameters enables/disables addresses if they have 0 tokens. After running the command, you should be able to see the following output.

```
[
  {
    "address": "tnLRVU32vCfGD6...",
    "amount": 0.00000000,
    "confirmations": 0,
    "label": "",
    "txids": [
    ]
  }
]
```

You can also use the `getwalletinfo` command to see aggregate information if you have generated multiple addresses.

```
./defi-cli -testnet getwalletinfo
```

```
{
  "walletname": "",
  "walletversion": 169900,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 4,
  "keypoololdest": 1646714869,
  "keypoolsize": 999,
  "keypoolsize_hd_internal": 1000,
  "paytxfee": 0.00000000,
  "hdseedid": "c43d012bb7c93cd9129dfc42b6643de774b2502f",
  "private_keys_enabled": true,
  "avoid_reuse": false,
  "scanning": false
}
```

Getting some testnet DFI

Creating a transaction

Once you have some DFI in your wallet, you use it to create a vault and mint dTokens, or send it to other addresses. There are two options to create a transaction, writing a raw transaction or using the P2PKH template. We will go over sending DFI to another address using the P2PKH template.

Verify your wallet balance using the `getbalance` command.

```
./defi-cli -testnet getbalance
```

```
39999.99999160
```

You can now use the `sendtoaddress` command to send DFI.

```
./defi-cli -testnet sendtoaddress tiiUqg4TqGR7ja... 1
```

The value after the address sets the amount of DFI to transfer. The command will output the new transaction ID.

```
23ee92302e166daf540d358fa726d52ff391f9f3058636ca45d887c747cd8610
```

You can inspect the transaction using the `gettransaction` or `getrawtransaction` command.

```
./defi-cli -testnet gettransaction
23ee92302e166daf540d358fa726d52ff391f9f3058636ca45d887c747cd8610
./defi-cli -testnet getrawtransaction
23ee92302e166daf540d358fa726d52ff391f9f3058636ca45d887c747cd8610 true
dbbe9b1e4f116d9316a9d1dc024fd4301b72e1f18ed1706fb59711315844a7ff
                                        ^transaction hash
^verbosity          ^block hash
```

Sample output for `gettransaction`

```
{
  "amount": 0.00000000,
  "fee": -0.00000840,
  "confirmations": 4,
  "blockhash":
"dbbe9b1e4f116d9316a9d1dc024fd4301b72e1f18ed1706fb59711315844a7ff",
  "blockindex": 1,
  "blocktime": 1647880878,
  "txid": "23ee92302e166daf540d358fa726d52ff391f9f3058636ca45d887c747cd8610",
  "walletconflicts": [
  ],
  "time": 1647880831,
  "timereceived": 1647880831,
  "bip125-replaceable": "no",
  "details": [
    {
      "address": "tnLRVU32vCfGD6BB5TywSBgHwre84WrZK8",
      "category": "send",
      "amount": -1.00000000,
      "label": "",
      "vout": 0,
      "fee": -0.00000840,
      "abandoned": false
    },
    {
      "address": "tnLRVU32vCfGD6BB5TywSBgHwre84WrZK8",
      "category": "receive",
      "amount": 1.00000000,
      "label": "",
      "vout": 0
    }
  ],
  "hex":
"04000000000101a421ebc69f999ea8db7e0322994965518a28e0ed109880bfd9b51a080242586a00
}
```


The other address should recieve the tokens as soon as the transaction is included in a block!

Security

Your wallet is **not encrypted by default**. This means that anyone with physical access to your computer or wallet file will be able to access your funds. For any wallets with real DFI or dTokens, make use of the `encryptwallet` command to encrypt your wallet, and use `walletlock` lock your wallet with a password and prevent hackers from accessing your funds.

Loans

Table of contents

1. Introduction

In this document we will try to understand how loans work inside DeFiChain and how they differ from traditional lending services in CeFi (Centralized Finance).

NOTE

We will be speaking a lot about **amount** and **value** when talking about dTokens, collaterals, loans and interests. It is important to make clear the difference between both.

Amount is the quantity of the referred asset and value is the conversion to $ taken from the oracles.

To make this more clear lets see an example:

> Given 10dTSLA with each tesla priced at 100$:
>
> Amount = 10
>
> Value = 10dTSLA * 100$ = 1000$

This documentation is in continuous development, feel free to open an issue or Pull Request to the official repository.

2. Loans in DeFiChain

To better understand the difference between DeFiChain loans and traditional loans lets start by looking at the definition of traditional *loan* by the Cambridge Dictionary:

> ---
>
> *Loan: An amount of money that is borrowed, often from a bank, and has to be paid back, usually together with an extra amount of money that you have to pay as a charge for borrowing*
>
> ---

Lets adjust this definition to fit in the DeFiChain's implementation of loans.

> *"an amount of money that is borrowed, often from bank, and has to be paid back...".*

When operating in DeFiChain it is more accurate to talk about loan tokens or dTokens instead of borrowed money as all operations regarding loans. Moreover, loan tokens are not borrowed like in traditional finance as there is no one lending the assets in the other side. Instead, loan tokens are minted, this is how we call the process to create new dTokens on the blockchain.

For a better understanding of dTokens please have a look at this blog post. For now lets say, *borrowed money*=loan tokens.

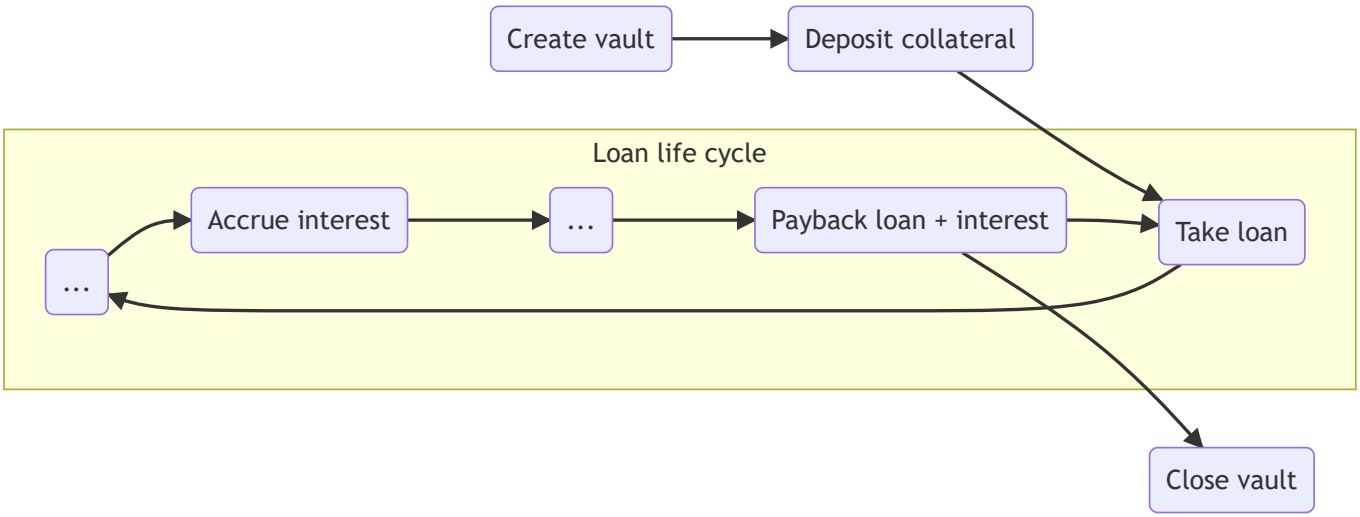
Continuing with the adaptation of the definition, in DeFi (Decentralized finance) there are no central authorities so we can remove any reference to banks. Again, the key part to understand here is that there is not one specific person nor entity approving or rejecting a loan, the network through consensus rules and algorithms will be in charge of providing users for their loans tokens.

The last part *"...and has to be paid back."* needs also some refactoring as in DeFiChain **loans have no due date**. In DeFiChain loans need to be backed up by collateral tokens value. We can spot a similar behaviour in traditional finance where the bank asks for a deposit of value as a guarantee (a house, a car...) in return of a certain amount of money.

Applying the changes mentioned, our definition of loan is now:

> An amount of loan tokens that is minted, ~~often from a bank~~, and which value has to be backed up by collateral during the loan's life cycle.

The loan's life cycle ends when a loan is fully paid back, i.e. all dTokens + interests are returned to the vault.



Now lets continue to the second part of the initial definition:

> *"...usually together with an extra amount of money that you have to pay as a charge for borrowing"*

This part talks about **interests** and of course DeFiChain loans also generate interests. Each vault subscribes to a loan scheme, this loan scheme sets the amount of interest that the minted tokens will generate. Also dTokens can be set to generate interests themselves regardless the vault they operate in. This is explained in detail in the interests definition.

Now we are finished with the adaptation of the definition to suit the DeFiChain implementation of loans, lets have a look at the final result:

> An amount of dToken that is minted and which value + interests has to be backed up by collateral during the loan's life cycle.

---

3. Main concepts

In this section you will find a detailed explanation of the concepts and key aspects regarding loans in DeFiChain.

Collateral

Collateral is the guarantee deposited into a vault that let vault owners mint dTokens in DeFiChain i.e., take a loan. This collateral will be locked in the vault until a withdraw or closevault operation succeeds.

Collateralization ratio

The collateralization ratio is the value of the deposit with respect to the value of the minted or loan tokens plus the interests.

It is calculated using the following formula:

```
Collateralization ratio =
    Total value of collateral /
    (Total value of tokens minted + Total value interest)
```

Over-collateralization

Vaults follow an over-collateralization mechanism, i.e., the value of the collateral locked in a vault must be of greater value than the one of the minted dTokens + the interests. How much a vault must be over-collateralized is defined by the minimum collateralization ratio. This is set by the loan scheme the vault subscribes to.

Collateralization factor

Assets that are accepted for collateralization could be accepted at different collateralization factors, from 0% to 100%. When an asset's collateralization factor is 100%, the entirety of the

asset's value contributes to the collateralization value of the vault, however, if an asset's collateralization factor is say 50%, then only half of its value contributes to the total vault's collateral.

For example if `DOGE` is accepted at 70% collateralization factor, $100 worth of `DOGE` would contribute to $70 of collateral value in a vault.

This is not to be confused with *collateralization ratio*.

Minting

Minting dTokens is the process by which dTokens are created in DeFiChain. dTokens in DeFiChain can be minted by creating a vault, depositing collateral and taking a loan. As vaults are over-collateralized, this process ensures that all dTokens are backed up by the collaterals locked in vaults at all times.

This loaned tokens can be freely used for other purposes like adding liquidity to a pool, swaping in a liquidity pool or transfer to another address.

Loan scheme

Loan schemes are structures that define the conditions vaults adhere to.

A loan scheme has the following information:

- An `Id` (String) used as a unique identifier of the loan scheme.
- A `minimum collateralization ratio` (Natural number greater or equal to 100)
- An `interest rate` (positive rational number from 0-100)

Lets see some examples of possible loan schemes:

- Id: "LOAN150". Min collateralization ratio: 150%. Interest rate: 5% APR.
- Id: "LOAN175". Min collateralization ratio: 175%. Interest rate: 3% APR.
- Id: "LOAN200". Min collateralization ratio: 200%. Interest rate: 2% APR.
- Id: "LOAN350". Min collateralization ratio: 350%. Interest rate: 1.5% APR.
- Id: "LOAN500". Min collateralization ratio: 500%. Interest rate: 1% APR.
- Id: "LOAN1000". Min collateralization ratio: 1000%. Interest rate: 0.5% APR.

Following the above list, a vault subscribed to "LOAN150" would need to have always a collateralization ratio over 150% and it's loans will generate a 5% interest per year.

Price and fixed interval price

Price in DeFiChain can be found in two forms:

- DEX price: obtained by the balance inside the liquidity pools.
- Oracle price: retrieved from the actual asset represented by the dToken outside DeFiChain.

Oracles are trusted services that inject this prices directly into DeFiChain. Read more about oracles here.

The important thing to understand here is that these prices are different and each is used for different purposes. For instance, when we talk about value, this is calculated with the oracle price and not with the DEX price. DEX price in the other hand is used when swapping between dTokens and DFI to burn the interests or the penalty generated by a vault being liquidated.

Fixed Interval Price

Is a mechanism to provide users time to prevent the liquidation of their vaults when the price of the collateral tokens or loan tokens is unstable. This is achieved by fixing the price of the tokens each 120 blocks (~1 hour) for vault calculations.

There are in reality three prices for each dToken all of them taken from the oracles:

- `Active price`: the price used for collateralization ratio calculation.
- `Current price`: this is the live price of the oracle used to calculate the dTokens that can be minted within a vault at the time of taking the loan.
- `Next price`: the next price that will be used for calculations, i.e. the next price that will become active.

Token price feeds become invalid in the following scenarios:

- Active price and next price deviation is over 30%
- Active price is 0.
- Next price is 0.
- There are less than two active oracles for that price feed.

Minimum collateralization ratio

This is one of the most important aspects to understand vaults functioning and to do a proper risk management of your loans. The minimum collateralization ratio sets a threshold

that will trigger the liquidation of a vault if crossed. Basically if collateralization ratio falls under this number, the collaterals locked will be publicly auctioned along with a penalty fee.

We will cover the liquidation process later but for now take a note to remember that **you want your vault's collateralization ratio to be always over the minimum collateralization ratio.**

Interest rate

Interest rate for loan consists of 2 parts:

- Vault interest, based on loan scheme of individual vaults.
- Token interest, based on loan tokens, e.g. `TSLA` token might have its own interest that is chargeable only for `TSLA` token.

DeFi fees are burned. Fees are typically collected in the form of the loan token repayment, and automatically swapped on DEX for DFI to be burned.

For example, if a loan of 100 `TSLA` is taken out and repaid back exactly 6 months later with the APR of 2%, the user will have to repay 101 `TSLA` during redemption to regain full access of collateral in the vault.

100 `TSLA` will be burned as part of the repayment process. 1 `TSLA` that forms the interest payment will be swapped on DEX for DFI. The resulting DFI will burned. All of these occur atomically as part of DeFiChain consensus.

Liquidation

Liquidation starts when the collateralization ratio of a vault falls below the minimum collateralization ratio set by the loan scheme a vault subscribes to. Liquidation is crucial in ensuring that all loan tokens are sufficiently over-collateralized to support its price.

For Turing-complete VM-based blockchain, liquidation requires the assistance of publicly run bots to trigger it, in exchange for some incentives.

There are two problems with that:

- During major price movements, the network fee, or commonly known as gas price, would sky rocket due to a race by both the vault owners frantically trying to save their vaults from getting liquidated, and by the public liquidators trying to trigger liquidation.
- If liquidations are not triggered on time, tokens that are generated as a result of loan runs the risk of losing its values. Worse scenario would see it getting into a negative feedback loop situation creating a rapid crash of token values.

DeFiChain, a blockchain that's built for specifically for DeFi, enjoy the benefit of automation and can have liquidation trigger automatically on-chain. This address the above two problems.

Liquidation penalty

If a vault gets liquidated then a penalty fee is charged. Default penalty is set to 5% of the total value of the minted dTokens including interest value.

Vault

A vault is the place where collaterals (guarantee) and loans (minted dTokens) are handled. It is also accountant of the generated interests over time.

Vaults are identified by a unique `Id` on the blockchain. This `Id` is the transaction hash of the creation of the vault, meaning the return of the `createvault` RPC command. To create an empty vault only a valid address is needed, nothing else. If a loan scheme is not set upon creation, the vault will be subscribed to the default loan scheme.

There are many actions that can be made regarding vaults. The following actions can only be performed by the owner of the vault:

- Update the vault (change owner address and/or loan scheme).
- Deposit collateral.
- Take a loan (mint dTokens).
- Withdraw collaterals from the vault.
- Close the vault.

Other actions that can be performed by anyone in the blockchain:

- Create a vault.
- Pay back a loan.

There are also actions that aim to retrieve vault information and help with the calculation estimation.

This commands can also be run by any node in the network:

- Get a vault's current information.
- List a vault's history (collateral ratios, actions, value changes, states...)
- Estimate loan. (return the maximum amount of dToken that can be minted with the current collateral value locked in a vault)
- Estimate collateral. (return the minimum amount of collateral needed under a certain loan scheme to mint a given number of dTokens)

- Estimate vault. (give back the estimation of collateral value, loan value and ratios that a vault would have given an amount of collaterals and an amount of minted tokens)

This is just an introduction to the several actions you can make around vaults. All this actions are translated into RPC commands and will be covered in depth in the RPC section of this paper.

Vault state

Depending on the status of the dTokens and the balance between the assets in a vault (collaterals, loans and interests), the vault will be subject to enter different states:

- `active`: collateralization ratio is over the minimum collateralization ratio and both collaterals and loans price feeds are valid.
- `frozen`: collateral or loan price feed is invalid.
- `inLiquidation`: collateralization ratio is below the minimum collateralization ratio thus vault is under liquidation and has undergoing auctions. When a vault enters in liquidation state, it can no longer be used until all the loan amount is repaid, only then the vault will be reopened. Interest accrual is also stopped during liquidation process.
- `mayLiquidate` : collateralization ratio calculated with next price is below minimum collateralization ratio, meaning the vault will liquidate at the next price update unless loan is paid back or collateral deposited.

Every time an action in the vault is triggered, the state is checked to allow or deny the execution of the action. This table represents the action/state relation:

Action/State	Active	Frozen	InLiquidation	MayLiquidate
Update	✅	❌	❌	✅
Take loan	✅	❌	❌	✅
Withdraw collateral	✅	❌	❌	✅
Close	✅	✅	❌	❌
Deposit collateral	✅	✅	❌	✅
Loan payback	✅	❌	❌	✅

*All other actions can occur regardless the state of the vault.

Now as you can see in the table nothing can be done once a vault enters the `inLiquidation` state, so what happens when a vault gets liquidated? DeFiChain starts the liquidation process and a public auction takes place.

We will explore the details of these auctions in the next section.

50% rule

This rule applies to vaults and particularly to collaterals. The rule states:

> At least 50% of the minimum required collateral must be in `DFI` or/and `DUSD`.

Example

Having a vault which has a minimum collateralization ratio of 100%. This means the minimum collateral needed would be equal to the value of the loan taken. i.e. If we have 1000$ of dTSLA we would need a minimum of 1000$ of collateral.

Following with the example, this rule forces the vault owner to have at least 50% of the 1000$ collateral either in DFI, DUSD or the sum of both. For instance, having 1000$ of dBTC as collateral would break this rule, we would need at least 500$ (DUSD+DFI) and 500$ in dBTC as collateral.

This rule only affects `takeloan` action. This means no loan can be taken in a vault that breaks the `50% rule`.

Auction

An auction is the mechanism implemented in DeFiChain to recover the dTokens that are at risk of not being backed up by collaterals in vaults. As we have seen, vaults are over-collateralized so this auctions are triggered way before the dTokens are at risk of not being backed up by the value of the collaterals.

Auction batch

The entirety of loan and collateral of a vault is put up for auction along with the `LOAN_LIQUIDATION_PENALTY`. As the first version of collateral auction requires bidding of the full amount, auction is automatically split into **batches** of around $10,000 worth each to facilitate for easier bidding.

A batch contains the relevant information for bidders:

- minimum bid amount of dToken
- amount of collaterals that the winner of the auction will receive.

Batches may contain multiple collateral tokens, but not more than one loan token.

For instance, a liquidating vault that consists of the following:

- Collateral: 1 dBTC & 100 DFI
- Loan: $10k worth of DUSD * $40k worth TSLA

would be liquidated as follows:

- (0.2 dBTC & 20 DFI) for $10k worth of DUSD
- (0.8 dBTC & 80 DFI) for $40k worth of dTSLA

and further split into $10k batches.

Example

A vault, that requires a minimum of 150% collateralization ratio contains $15,000 worth of collateral, which consists of $10,000 worth of DFI and $5,000 worth of dBTC, and the total loan, inclusive of interest, is $11,000 worth.

- Collateral: $10,000 DFI + $5,000 dBTC
- Loan: $10,000 `dTSLA` + $1,000 interest (`dTSLA`)

Collateralization ratio = 15k / 11k = 136.36% less than the required 150%. Liquidation is triggered automatically by consensus. Auction will be initiated with the intention to liquidate $15k of collateral to recover $11k of loan.

Liquidation penalty is defined as `LOAN_LIQUIDATION_PENALTY`. The amount to be recovered should also include the penalty. If `LOAN_LIQUIDATION_PENALTY` is 0.05, the total amount to be recovered is thus $11k * 1.05 = $11,550

Total collateral worth is $15k, it will thus be split into 2 batches, each not exceeding $10k:

1. 2/3 of the whole vault, worth $10k of collateral.
2. 1/3 of the whole vault, worth $5k of collateral.

Specifically the following:

1. ($6,667 DFI and $3,333 dBTC) for $7,700 `dTSLA`
2. ($3,333 DFI and $1,667 dBTC) for $3,850 `dTSLA`

`dTSLA` recovered at the conclusion of the auction will be paid back to the vault. Once all loan is repaid, vault will exit liquidation state, allowing its owner to continue to use it. It will, however, not terminate all opened auctions. Opened auctions see through to their conclusion.

Collateral auction example

Collateral auction will run for 720 blocks (approximately 6 hours) with the starting price based on liquidation vault's ratio.

Using the same illustration, Auction 1: ($6,667 DFI and $3,333 dBTC) for $7,700 `dTSLA`

Anyone can participate in the auction, including the vault owner by bidding for the entirety of the auction with higher amount. To prevent unnecessary auction sniping at closing blocks, each higher bid, except the first one, has to be at least 1% higher than previous bid.

A user could bid for the same auction by offering $7,800 worth of `dTSLA` tokens. During bidding, the bid is locked and refunded immediately when outbid. Top bid is not cancelable.

At the conclusion of the auction, the entirety of the for sale collateral will be transferred to the winner – in this case the winner would be paying $7,800 for $10,000 worth of BTC and DFI.

$7,800 worth of `dTSLA` token will be processed as follow:

- Paid back to vault = $7,800 worth of `dTSLA` / (1 + `LOAN_LIQUIDATION_PENALTY`) = 7800 / 1.05 = $7,428.57 `dTSLA` token
- The remaining, i.e. liquidation penalty, will be swapped to DFI and burned. $371.43 worth of `dTSLA` be swapped as follows and trackably burned:
    1. `dTSLA` to `DUSD` from `dTSLA-DUSD` DEX.
    2. `DUSD` to `DFI` from `DUSD-DFI` DEX.

If auctions yield more `dTSLA` than the vault's loan, it will be deposited back to the vault as part of vault's asset, but not collateral. It can be withdrawn from the vault after vault exits liquidation state by vault's owner and carries no interest.

Failed auction

Auction that expires after 720 blocks without bids will be reopened immediately based on the remaining amount of collateral and loan that the vault receives due to conclusion of other auction batches.

4. RPC

Now we will go through all the different RPCs implemented in DeFiChain to handle loans.

Note: In this section a red asterisk * means the parameter is required. If no asterisk is

present the argument will then be optional.

Loan scheme RPCs

- `createloanscheme`:

  Is used to create a loan scheme. The first loan scheme created will be set as the default loan scheme.

  Arguments:

  - `mincolratio`*: e.g. 175 for 175% minimum collateralization ratio. Cannot be less than 100.
  - `interest rate`*: annual interest rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be > 0.
  - `id`*: non-colliding scheme ID that's unique within opspace, e.g. MIN_175

  Return: Transaction hash.

  Example usage:

  ```
  defi-cli createloanscheme 150 5 LOAN150 # mincolratio 150%, interest rate 5%, id "LOAN150"
  ```

- `updateloanscheme`:

  Used to update a loan scheme.

  Arguments:

  - `mincolratio`*: e.g. 175 for 175% minimum collateralization ratio. Cannot be less than 100.
  - `interest rate`*: annual interest rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be > 0.
  - `id`*: id of loan scheme that will be updated
  - `ACTIVATE_AFTER_BLOCK`: if set, this will only be activated after the set block. The purpose is to allow to provide sufficient warning.

  Return: Transaction hash.

  Example usage:

  ```
  defi-cli createloanscheme 150 5 LOAN150
  defi-cli createloanscheme 150 5 LOAN150 3020 # Update only after block 3020
  ```

- `destroyloanscheme`:

  Used to destroy a loan scheme.

  Arguments:

  - `id`*: the unique identifier of the loan scheme to destroy.
  - `ACTIVATE_AFTER_BLOCK`: if set, loan will be destroyed after the set block. The purpose is to allow to provide sufficient warning.

  Return: Transaction hash.

  Example usage:

  ```
  defi-cli destroyloanscheme LOAN150
  defi-cli destroyloanscheme LOAN150 3020 # Destroy only after block 3020
  ```

  Note

  - All vaults under this loan scheme will be moved to the default loan scheme after destruction.
  - Default loan scheme cannot be destroyed.

- `setdefaultloanscheme`:

  Used to change the default loan scheme.

  Arguments:

  - `id`*: the unique identifier of the new default loan scheme.

  Return: Transaction hash.

  Example usage:

  ```
  defi-cli setdefaultloanscheme LOAN150
  ```

- `getloanscheme`:

  Get loan scheme information.

  Arguments:

  - `id`*: unique identifier of a loan scheme.

  Return:

```
{
  "id": "C150",
  "mincolratio": 150,
  "interestrate": 5.00000000,
  "default": true
}
```

Example usage:

```
defi-cli getloanscheme C150
```

- `listloanschemes`:

  Retrieve the list of all loan schemes. This command receives no arguments.

  Return:

```
[
      {
        "id": "C150",
        "mincolratio": 150,
        "interestrate": 5.00000000,
        "default": true
      },
      {
        "id": "C175",
        "mincolratio": 175,
        "interestrate": 3.00000000,
        "default": false
      },
      .
      .
      .
]
```

  If there are no loan schemes an empty list is returned []

  Example usage:

```
defi-cli listloanschemes
```

Loan tokens RPCs

- `setloantoken`:

  Creates a loan token.

  This arguments are wrapped in a `metadata` object see the example:

  - `symbol`*: token's symbol (unique).
  - `fixedIntervalPriceId`*: the id of the price feed to track tokens price
  - `name`: token's name. Default is set to an empty string.
  - `mintble`: token's mintable property. Defaults to True.
  - `interest`: annual interest rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be > 0. Defaults to 0.

  Return: Transaction hash.

  Example usage:

```
defi-cli setloantoken {"symbol":"TSLA", \
                       "name":"TSLA stock token", \
                       "fixedIntervalPriceId":"TSLA/USD", \
                       "interest":"3"}
```

- `updateloantoken`:

  Updates a loan token.

  - `token`*: token's symbol, id or creation transaction hash.
  - `metadata`*: object containing the data to be updated.
    - `symbol`: token's symbol (unique).
    - `fixedIntervalPriceId`: the id of the price feed to track tokens price
    - `name`: token's name. Default is set to an empty string.
    - `mintble`: token's mintable property. Defaults to True.
    - `interest`: annual interest rate, but chargeable per block (scaled to 30-sec block). e.g. 3.5 for 3.5% interest rate. Must be > 0. Defaults to 0.

  Return: Transaction hash.

  Example usage:

```
defi-cli updateloantoken TSLA {"symbol":"NEWTSLA", \
                               "name":"NEWTSLA stock token", \
                               "fixedIntervalPriceId":"NEWTSLA/USD", \
                               "interest":"1"}
```

- `getloantoken`:

  Get relevant information about a loan token.

Arguments:

- `token`*: token's symbol, id or creation transaction hash.

Return: There are only two attributes differencing a token from a loan token: `interest` and `fixedIntervalPricedId` which is for tracking tokens oracle price. Lets see an example of real return for a loan token:


```
{
    "token": {
        "19": {
            "symbol": "TSLA",
            "symbolKey": "TSLA",
            "name": "TSLA",
            "decimal": 8,
            "limit": 0,
            "mintable": true,
            "tradeable": true,
            "isDAT": true,
            "isLPS": false,
            "finalized": false,
            "isLoanToken": true,
            "minted": 40018.13292270,
            "creationTx":
"ea485478eb87885cd083faf0dfacd9253d9f15d9bdc016ebacb50becb47e4ce6",
            "creationHeight": 686335,
            "destructionTx":
"0000000000000000000000000000000000000000000000000000000000000000",
            "destructionHeight": -1,
            "collateralAddress": "7Q2nZCcKnxiRiHSNQtLB27RA5efxm2cE7w"
        }
    },
    "fixedIntervalPriceId": "TSLA/USD",
    "interest": 6.00000000
}
```


If loan token is not found an empty object is returned {}

Example usage:

```
defi-cli getloantoken TSLA # symbol
defi-cli getloantoken 0    # token id
defi-cli getloantoken
c207f70156adb13a76d3ae08814d6735670b901079cbae5e4f1b5ca7bb9e2573 # tx
creation hash
```

- `listloantokens`:

  Retrieve the list of all loan tokens. This command receives no arguments.

  Return:

```
[
      {
            "token": {
                  "10": {
                    "symbol": "TWTR",
                    "symbolKey": "TWTR",
                    "name": "TWTR",
                    "decimal": 8,
                    "limit": 0,
                    "mintable": true,
                    "tradeable": true,
                    "isDAT": true,
                    "isLPS": false,
                    "finalized": false,
                    "isLoanToken": true,
                    "minted": 40492.38394616,
                    "creationTx":
"35113ddb109e60867d1c8a5eddcee4ee73f1f704954a89ab7d0ec12247ec2d4d",
                    "creationHeight": 686314,
                    "destructionTx":
"0000000000000000000000000000000000000000000000000000000000000000",
                    "destructionHeight": -1,
                    "collateralAddress":
"7Q2nZCcKnxiRiHSNQtLB27RA5efxm2cE7w"
                  }
            },
            "fixedIntervalPriceId": "TWTR/USD",
            "interest": 2.10000000
      },
      {
            "token": {
                  "11": {
                    "symbol": "DUSD",
                    "symbolKey": "DUSD",
                    "name": "DUSD",
                    "decimal": 8,
                    "limit": 0,
                    "mintable": true,
                    "tradeable": true,
                    "isDAT": true,
                    "isLPS": false,
                    "finalized": false,
                    "isLoanToken": true,
                    "minted": 5177661.38853472,
```

```
                    "creationTx":
"cac1cd0a39e8ac2351ac68075fa651bd831eeef25b1dee005928fdbb2c431d5e",
                    "creationHeight": 686314,
                    "destructionTx":
"0000000000000000000000000000000000000000000000000000000000000000",
                    "destructionHeight": -1,
                    "collateralAddress":
"7Q2nZCcKnxiRiHSNQtLB27RA5efxm2cE7w"
                }
            },
            "fixedIntervalPriceId": "DUSD/USD",
            "interest": 1.00000000
      },
      .
      .
      .
]
```

If there are no loan tokens an empty list is returned []

Example usage:

```
defi-cli listloantokens
```

Collateral tokens RPCs

- `setcollateraltoken`:

  Creates updates and destroys a collateral token.

  This arguments are wrapped in a `metadata` object see the example:

  - `token`*: token's symbol or id of collateral token (unique).
  - `factor`*: collateral factor.
  - `fixedIntervalPriceId`*: token/currency pair to use for price of token.
  - `activeAfterBlock`: token will be active after block height. If not set it will be active immediately.

  Return: Transaction hash.

  Example usage:

```
defi-cli setcollateraltoken {"token": "TSLA", \
                            "factor": 1, \
                            "fixedIntervalPriceId": "TSLA/USD", \
                            "activeAfterBlock": 3020}
```

Note

This same command can be used to update an existing collateral token.

To destroy a collateral token just set collateral factor to 0.

- `getcollateraltoken`:

  Get relevant information about a collateral token.

  Arguments:

  - `token`*: token's symbol e.g. DUSD, DFI...

  Return:

```
{
  "token": "DUSD",
  "tokenId":
"d9fdcb425a8e770e8689a2cb04e3891b30031b511492981446ecaf8e5d479455",
  "factor": 0.99000000,
  "fixedIntervalPriceId": "DUSD/USD",
  "activateAfterBlock": 854986
}
```

  Example usage:

```
defi-cli getcollateraltoken BTC
```

- `listcollateraltokens`:

  Retrieve the list of all collateral tokens. This command receives no arguments.

  Return:

```
[
      {
        "token": "DFI",
        "tokenId":
"897dcc7567cd883d9e77e06ee786856db3708c27b75da936e55072fc5300ba89",
        "factor": 1.00000000,
        "fixedIntervalPriceId": "DFI/USD",
        "activateAfterBlock": 686335
      },
      {
        "token": "BTC",
        "tokenId":
"7705ba258dc7c2a7fb8a8e55d65726d83fbeadcabd9b857f29527dbe4da46f99",
        "factor": 1.00000000,
        "fixedIntervalPriceId": "BTC/USD",
        "activateAfterBlock": 686308
      },
      .
      .
      .
]
```

If there are no collateral tokens an empty list is returned []

Example usage:

```
defi-cli listcollateraltokens
```

Vault RPCs

- `createvault`:

  Is used to create an empty vault.

  Arguments:

  - `ownerAddress`*: the address owning this vault. This address might be owned by the node executing the command or not, meaning, anyone can create vaults to other users.
  - `loanSchemeId`: id of the loan scheme this vault is subscribed to. This defines the rules applied to this particular vault; the interest rate and the minimum collateral ratio. If omitted the vault will subscribe to the default loan scheme.

Return: `vaultId` (transaction hash). This can be used to identify the created vault inside the blockchain.

> **Note**
>
> 2 DFIs are charged when creating a vault:
>
> - 1 DFI will be burned.
> - 1 DFI will be added as collateral of the vault. This last 1 DFI will be send to the address indicated in `closevault` RPC command.
>
> So the overall cost of creating a vault is 1 DFI.

Example usage:

```
defi-cli createvault mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF LOAN150
defi-cli createvault mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF #  Choose default loan scheme
```

- `updatevault`:

Is used to update a vault's owner, loan scheme or both.

Arguments:

- `vaultId`*: id of the vault to be updated
- `parameters`*: object containing the attributes to be updated
  - `ownerAddress`: address to which transfer the vault.
  - `loanSchemeId`: id of the loan scheme this vault will be subscribed to.

Return: Transaction hash.

> **Note**
>
> - Requires ownerAddress authentication.
> - Update cannot happen while the vault is in either `inLiquidation` or `frozen` state.
> - If the loan scheme is set to be destroyed in the future with `activateAfterBlock`,the update of the vault to that loan scheme will not be allowed.

Example usage:

```
defi-cli updatevault
84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2 \
                  { "ownerAddress": mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF, \
                    "loanSchemeId": "LOAN150" }
```

- `closevault`:

  Is used to close an empty vault. Only vaults with no active loans can be closed by its owner.

  Arguments:

  - `vaultId`*: identifier of the vault to be closed.
  - `to`*: address to which transfer remaining collaterals (if any).

  Return: Transaction hash.

  Example usage:

  ```
  defi-cli closevault mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF LOAN150
  ```

- `getvault`:

  Get vault relevant information.

  Arguments:

  - `vaultId`*: identifier of the vault.

  Return:

```
{
  "vaultId":
"279e8a0dc5aa5c57baae66277d0135231934b7be7d8fcc2572ee703b6269eea5",
  "loanSchemeId": "C200",
  "ownerAddress": "tpsc2JrxHBEsRJ1PvXSq7pP12jDzBGydTT",
  "state": "active",
  "collateralAmounts": [
    "700.00000000@DFI"
  ],
  "loanAmounts": [
    "0.51774842@TSLA"
  ],
  "interestAmounts": [
    "0.00774842@TSLA"
  ],
  "collateralValue": 2199.30307800,
  "loanValue": 505.27742871,
  "interestValue": 7.56178403,
  "informativeRatio": 435.26644038,
  "collateralRatio": 435
}
```


Example usage:

```
defi-cli getvault
279e8a0dc5aa5c57baae66277d0135231934b7be7d8fcc2572ee703b6269eea5
```

- `listvaults`:

  Get the list of vaults on the network.

  Arguments:

  We have two optional parameters that will help with the **filtering** and **pagination** of the resulting list.

  - `options`: this object contains the parameters to filter the results
    - `ownerAddress`: return only vaults owned by this address.
    - `loanSchemeId`: return vaults that subscribe to a specific loan scheme.
    - `state`: retrieve vaults that are currently in a specific state (active, froze, inLiquidation...)
    - `verbose`: show the full information of the vaults listed. See the example returns to see the difference. (boolean)
  - `pagination`: this object contains the parameters needed for pagination purposes.

- `start`: optional first key to iterate from, in lexicographical order. Typically it's set to last ID from previous request.
- `including_start`: if true, then iterate including starting position. False by default.
- `limit`: maximum number of orders to return, 100 by default.

Return:

Verbose parameter not set or set to false:


```
[
      {
        "vaultId":
"d5611d2823b09eaa9c3a990ae2b372f6fa8c71cfce272c6f942d25f715732f01",
        "ownerAddress": "tXu6Q9UKN4rKEHykANSMSAQ6Gowijear2i",
        "loanSchemeId": "C150",
        "state": "active"
      },
      {
        "vaultId":
"6a63ec6a5f5fd00d1669e17ed547f54f5c561c10cfb1fadcee427688b5c34702",
        "ownerAddress": "tkjZa5HmeCVzry2Tq77tFouKjm5vQ3NsY7",
        "loanSchemeId": "C150",
        "state": "active"
      },
      .
      .
      .
]
```


Verbose parameter set to true:

```
[
      {
        "vaultId":
"d5611d2823b09eaa9c3a990ae2b372f6fa8c71cfce272c6f942d25f715732f01",
        "loanSchemeId": "C150",
        "ownerAddress": "tXu6Q9UKN4rKEHykANSMSAQ6Gowijear2i",
        "state": "active",
        "collateralAmounts": [
        ],
        "loanAmounts": [
        ],
        "interestAmounts": [
        ],
        "collateralValue": 0.00000000,
        "loanValue": 0.00000000,
        "interestValue": 0,
        "informativeRatio": -1.00000000,
        "collateralRatio": -1
      },
      .
      .
      .
]
```

Example usage:

```
defi-cli listvaults
defi-cli listvaults '{"loanSchemeId": "LOAN1502"}'
defi-cli listvaults '{"verbose": "true"}'
defi-cli listvaults '{"loanSchemeId": "LOAN1502"}'
'{"start":"3ef9fd5bd1d0ce94751e6286710051361e8ef8fac43cca9cb22397bf0d17e013",
"including_start": true, "limit":100}'
defi-cli listvaults {}
'{"start":"3ef9fd5bd1d0ce94751e6286710051361e8ef8fac43cca9cb22397bf0d17e013",
"including_start": true, "limit":100}'
```

- `listvaulthistory`:

  Get the history of a vault.

  Arguments:

  - `vaultId`*: identifier of the vault.
  - `options`: this object contains the parameters to filter the results.

- `maxBlockHeight`: optional height to iterate from (down to genesis block), (default = chaintip).
- `depth`: maximum depth, from the genesis block is the default.
- `token`: filter by token.
- `txtype`: filter by transaction type, supported letter from {CustomTxType} e.g. "CloseVault", "PaybackLoan", "UpdateVault"
- `limit`: maximum number of records to return, 100 by default.

Return:

```
[
      {
          'address': 'mqYr2o5sGRkBuUm7CkWiBWkM11naYMscog',
          'amounts': ['10.00000000@DFI'],
          'blockHash':
'29097a94dbeeac54de137eb706831062ef8296a402388fd212276d0b0d1945c3',
          'blockHeight': 127,
          'blockTime': 1649765301,
          'txid':
'e6b9b0aed8a0752547e8880685db559ea0abdb3648303fa15117fe02caa459f2',
          'txn': 1,
          'type': 'CloseVault'
      },
      {
          'address': 'mqkdzzVL36dsf9vABwWoq5iBjptS5vyUDU',
          'amounts': ['0.50000000@DFI'],
          'blockHash':
'7c8a41660884dc28e6e59af39ac662208dabc3f61e76b7a096e9d7953bd32804',
          'blockHeight': 126,
          'blockTime': 1649765300,
          'txid':
'a35ee0798c76c48da748f38b47c9abb945586473816308d06731eb352477084a',
          'txn': 1,
          'type': 'WithdrawFromVault'
      },
      {
          'address': 'mfburnZSAM7Gs1hpDeNaMotJXSGA7edosG',
          'amounts': ['0.00000228@DFI'],
          'blockHash':
'fa20e13c42d2dc3243efcb96f95e2953e30974b46191da074e271077437f2197',
          'blockHeight': 125,
          'blockTime': 1649765299,
          'txid':
'611da4b4204eb758f215a383f5602f048cc607f06e9dcd01aca087c963dc1131',
          'txn': 1,
          'type': 'PaybackLoan'
      },
      .
      .
      .
]
```

Example usage:

```
defi-cli listvaulthistory
84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2 \
                          '{"maxBlockHeight":160,"depth":10}'
```

- `deposittovault`:

  Is used to deposit collateral tokens in a vault. Only the owner of the vault can perform this action.

  Arguments:

  - `vaultId`*: identifier of the vault to which deposit the collateral amount.
  - `from`*: address account containing the amount of collateral.
  - `amount`*: amount of collateral in amount@symbol format. e.g. 1.3@dTSLA

  Return: Transaction hash.

  Example usage:

```
defi-cli deposittovault
84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2i
mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF 1.3@dTSLA
```

- `takeloan`:

  Is used to take a loan (mint dTokens).

  All arguments are wrapped in a metadata object, see example usage:

  - `vaultId`*: identifier of the vault to which deposit the collateral amount.
  - `to`: address to transfer minted tokens. If not specified a valid address will be taken from the wallet.
  - `amounts`*: amount of requested tokens in amount@symbol format. e.g. 20@dTSLA

  Return: Transaction hash.

  Example usage:

```
defi-cli takeloan '{"vaultId":
"84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2", \
                    "amounts": "10@TSLA"}'

defi-cli takeloan '{"vaultId":
"84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2", \
                    "to": "mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF", \
                    "amounts": "10@TSLA"}'
```

Note

Remember that to be able to take a loan the vault must comply the 50% rule.

- `paybackloan`:

Is used to payback loan tokens and interests. It does not require authentication so anyone can payback anyone's loan.

All arguments are wrapped in a metadata object, see example usage:

- `vaultId`*: identifier of the vault to which deposit the collateral amount.
- `from`*: address account containing the amount of collateral.
- `amounts`**: amount of collateral in amount@symbol format. e.g. 1.3@dTSLA
- `loans`**: object list containing specific payment for each loan token.
  - `dToken`*: dToken symbol to be paid.
  - `loans`*: amount in amount@token format that will pay for the above dToken

Return: Transaction hash.

Example usage:

```
# Using amounts
defi-cli paybackloan '{"vaultId":
"84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2i" \
                      "from": "mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF" \
                      "amounts": "1.3@dTSLA"}'

# Using loans object list
defi-cli paybackloan '{"vaultId":
"84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2i" \
                      "from": mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF \
                      "loans": [{"dToken": "dTSLA", "amounts": "1@DFI"} \
                                {"dToken": "dDOGE", "amounts":
"1@DUSD"}]}'
```

Note

** Either use `amounts` or `loans` but not both at the same time. The use of one of this arguments is mandatory.

Using the `loan` object allows a more specific way to pay back different loan tokens. If in the other hand `amounts` argument is used, the loan will be first paid back with the available DFI. If there is not enough balance, the rest of the loan will remain unpaid.

- `withdrawfromvault`:

Is used to take a loan (mint dTokens).

All arguments are wrapped in a metadata object, see example usage:

  - `vaultId`*: identifier of the vault from which withdraw the collateral amount.
  - `to`*: address to transfer collateral tokens.
  - `amounts`*: amount of withdrawn tokens in amount@symbol format. e.g. 20@dTSLA

Return: Transaction hash.

Example usage:

```
defi-cli withdrawfromvault '{"vaultId":
"84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2", \
                            "to": "mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF", \
                            "amounts": "14@TSLA"}'
```

Auction RPCs

- `listauctions`:

  Retrieve the list of all active auctions.

  Arguments:

  - `pagination`: this object contains the parameters needed for pagination purposes.
    - `start`: object providing `vaultId` and/or `liquidation height`{}
      - `vaultId`: id of the vault to start from.
      - `height`: start with auctions with `liquidationHeight` after this height
    - `including_start`: if true, then iterate including starting position. False by default.
    - `limit`: maximum number of orders to return, 100 by default.

  Return:

```
[
    {
         'batchCount': 1,
         'batches': [{'collaterals': ['100.00000000@DFI',
'100.00000000@BTC'],
                     'index': 0,
                     'loan': '10.00004560@TSLA'}],
         'liquidationHeight': 474,
         'liquidationPenalty': Decimal('5.00000000'),
         'loanSchemeId': 'LOAN0001',
         'ownerAddress': 'mwsZw8nF7pKxWH8eoKL9tPxTpaFkz7QeLU',
         'state': 'inLiquidation',
         'vaultId':
'415593c1803370cb83e6646357c9d28d1b890974b8f9934de2ac582272d0f027'
    },
    {
         'batchCount': 1,
         'batches': [{'collaterals': ['100.00000000@DFI',
'100.00000000@BTC'],
                      'index': 0,
                      'loan': '10.00004560@TSLA'}],
         'liquidationHeight': 474,
         'liquidationPenalty': Decimal('5.00000000'),
         'loanSchemeId': 'LOAN0001',
         'ownerAddress': 'mwsZw8nF7pKxWH8eoKL9tPxTpaFkz7QeLU',
         'state': 'inLiquidation',
         'vaultId':
'7db5032d699ac6696a9cc36a52b642f5785ca187c360116615d3855d0f760e61'
    },
    .
    .
    .
]
```

Example usage:

```
defi-cli listauctions

# With pagination
defi-cli listauctions {"start":
{"vaultId":"eeea650e5de30b77d17e3907204d200dfa4996e5c4d48b000ae8e70078fe7542"
\
                                    "height": 1000}, \
                          "including_start": true, \
                          "limit":100}
```

- `listauctionhistory`:

  Retrieve auction history.

  Arguments:

  - `owner|vaultId`: this parameter admits the following:
    - Single account id (CScript or address).
    - A vaultId.
    - One of this reserved words; `mine` - to list history for all owned accounts; `all` to list whole DB (default = `mine`).
  - `pagination`: this object contains the parameters needed for pagination purposes.
    - `maxBlockHeight`: height to iterate from (down to genesis block).
    - `vaultId`: show only auctions related to this vault.
    - `index`: only batches with this index will be listed.
    - `limit`: maximum number of orders to return, 100 by default.

  Return:

```
[
    {
        'auctionBid': '259.00000000@TSLA',
        'auctionWon': ['200.00000400@DFI', '200.00000400@BTC'],
        'batchIndex': 0,
        'blockHash':
'833f3b22f8df9b4d6d2e142790dd4a9f4e3b21b524e7f5303e5ea6396745671f',
        'blockHeight': 506,
        'blockTime': 1649803226,
        'vaultId':
'd1b414827c115160ed9eed11ec0bdfa4355f339ed5d6240974eeafe8867af0c2',
        'winner': 'mwsZw8nF7pKxWH8eoKL9tPxTpaFkz7QeLU'
    },
    {
        'auctionBid': '515.00000000@TSLA',
        'auctionWon': ['399.99999600@DFI', '399.99999600@BTC'],
        'batchIndex': 0,
        'blockHash':
'58582b7511253f8d1be20c0fad4358141f77255b6e2fa071121ac85cfbb7cbb2',
        'blockHeight': 469,
        'blockTime': 1649803189,
        'vaultId':
'd1b414827c115160ed9eed11ec0bdfa4355f339ed5d6240974eeafe8867af0c2',
        'winner': 'mwsZw8nF7pKxWH8eoKL9tPxTpaFkz7QeLU'
    },
    .
    .
    .
]
```

Example usage:

```
defi-cli listauctionhistory

# With pagination
defi-cli listauctionhistory {"maxBlockHeight": 1000, \

"vaultId":"eeea650e5de30b77d17e3907204d200dfa4996e5c4d48b000ae8e70078fe7542",
\
                            "index": 1, \
                            "limit":100}
```

- `placeauctionbid`:

Make bid for an open auction's batch.

Arguments:

- `vaultId`*: identifier of the vault in liquidation.
- `index`*: index of the target batch in the auction.
- `from`*: address to get tokens from. If set to `*` (star), an account with sufficient funds is selected from wallet.
- `amounts`*: amount of bid in amount@symbol format. e.g. 20@dTSLA

Return: Transaction hash.

Example usage:

```
defi-cli placeauctionbid
84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2 \
                          0 \ # batch index
                          mwSDMvn1Hoc8DsoB7AkLv7nxdrf5Ja4jsF \
                          100@TSLA
```

Helper RPCs

- `estimateloan`:

  Returns amount of loan tokens a vault can take depending on a target collateral ratio.

  Arguments:

  - `vaultId`*: identifier of the vault in liquidation.
  - `tokens`*: object with the loan tokens as key and their percent split as value. e.g. {"dTSLA": 0.1, "GOOGL" 0.9}
  - `targetRatio`: target collateral ratio. (defaults to vault's loan scheme ratio)

  Return: Array of amount@token e.g. ['1.5@TSLA', '2.2@TWTR']

  Example usage:

```
defi-cli estimateloan
84b22eee1964768304e624c416f29a91d78a01dc5e8e12db26bdac0670c67bb2 \
                       '{"TSLA":0.8, "TWTR": 0.2}'
```

- `estimatecollateral`:

  Returns amount of loan tokens a vault can take depending on a target collateral ratio.

  Arguments:

- `loanAmounts`*: loan amount for the estimation.
- `targetRatio`*: target collateral ratio. (defaults to vault's loan scheme ratio)
- `tokens`: object with the loan tokens as key and their percent split as value. e.g. {"dTSLA": 0.1, "GOOGL" 0.9}

Return: Array of amount@token e.g. ['5@TSLA', '4.01@TWTR']

Example usage:

```
defi-cli estimatecollateral 23.55311144@MSFT \
                            150 \
                            '{"DFI": 0.8, "BTC":0.2}'
```

- `estimatevault`:

  Returns estimated vault for given collateral and loan amounts.

  Arguments:

  - `collateralAmounts`*: collateral amount as JSON string or array e.g. ["amount@token"].
  - `loanAmounts`*: loan amount as JSON string or array e.g. ["amount@token"].

  Return:

```
{
    "collateralValue": 3134.14508000, # in USD
    "loanValue": 1685.77665785, # in USD
    "informativeRatio": 185.91698167,
    "collateralRatio": 186
}
```

  Example usage:

```
defi-cli estimatevault '["1000.00000000@DFI"]' '["0.65999990@GOOGL"]'
```

- `getloaninfo`:

  Returns the attributes of loans offered by Operator.

  Return:

```
{
    "currentPriceBlock": 928920,
    "nextPriceBlock": 929040,
    "defaults": {
        "scheme": "C150",
        "maxPriceDeviationPct": 30.00000000,
        "minOraclesPerPrice": 2,
        "fixedIntervalBlocks": 120
    },
    "totals": {
        "schemes": 6,
        "collateralTokens": 4,
        "collateralValue": 19892664.24475404,
        "loanTokens": 6,
        "loanValue": 501988.43640161,
        "openVaults": 163,
        "openAuctions": 0
    }
}
```


Example usage:

```
defi-cli getloaninfo
```

Decentralised Exchange

Table of contents

Introduction

The DeFiChain Decentralised Exchange is a marketplace where users can exchange their tokens for other on chain tokens. The DeFiChain DEX uses the automated market maker model where asset prices are defined by the ratio of available liquidity in the liquidity pool.

Each DeX pair (colloquially called pool pair) has the following properties:

Name	Description
Token A ID	Token ID of the first asset in the pool
Token B ID	Token ID of the second asset in the pool (generally `0` for DFI)
A reserves	Amount of Token A available in pool pair
B reserves	Amount of Token B available in pool pair
Commission	Swap fees for pool pair

The price of an asset can be determined by the ratio between the reserves of the two assets in the pool pair.

> In the DeFiChain DEX, liquidity is distributed uniformly.
>
> The DEX uses the `x * y = k` formula. Learn more about how it works in the Uniswap paper on the model and this article.

$$price_A = \frac{reserve_B}{reserve_A}$$

$$price_B = \frac{reserve_A}{reserve_B}$$

The formula used for calculating output amount when swapping from A to B is

$$output = reserve_B - \frac{reserve_B * reserve_A}{reserve_A + input}$$

Swaps are executed with the following steps:

1. Check if pool pair is activated
2. Check if minimum liquidity is available in the pool pair
3. Check if price is under maximum price
4. Subtract swap fees from input amount
5. Subtract DeX input fees from input amount (set using governance variables)
6. Add input amount to reserves and extract output from reserves

There are two possible types of swaps, **single** and **composite**. Single swaps are swaps where the swap route has just one pool pair, whereas composite swaps can go through up to 3 pool pairs.

For example, swapping from ETH to DFI is a single swap since it only uses the ETH-DFI pool. However swapping from DUSD to ETH is a composite swap since the DUSD is first swapped to DFI using the DUSD-DFI pool, and then the DFI is swapped to ETH using the ETH-DFI pool.

When using composite swaps, every pool pair used will apply their own fees. See FAQ.

See how the amount of LP tokens are caluclated here.

LP Rewards

Pools have block rewards and swap fees (called commissions). These rewards are paid out in proportion to the user's share of the total liquidity. LP rewards are transferred to the user when performing interactions any pool, such as adding/removing liquidity and swapping.

$$share = \frac{liquidity}{totalLiquidity}$$

Depending on the token type, the block reward is a share of the "Liquidity Pools" coinbase allocation (`rewardPct`) for crypto tokens, or "Loans" (`rewardLoanPct`) for mintable dTokens.

Performing swaps

Swapping tokens can be performed using the `poolswap` RPC for single swaps and `compositeswap` RPC for composite swaps.

The `testpoolswap` RPC can be used to simulate the results of the pool swap. Use the path argument to force simple or composite swaps.

Single swaps

To perform single swaps, the following metadata must be provided to the RPC endpoint

- from: address which sends input tokens
- tokenFrom: ID or symbol of input token
- amountFrom: amount of tokens to swap
- to: address to send output to
- tokenTo: ID or symbol of output token
- maxPrice: maximum acceptable price

Optionally, users can specify UTXOs to spend for the swap.

The metadata and UTXO data is passed as a JSON object to the RPC.

```
defi-cli poolswap
{"from":"str","tokenFrom":"str","amountFrom":n,"to":"str","tokenTo":"str","maxPri
( [{"txid":"hex","vout":n},...] )
defi-cli poolswap '{"from": "MyAddress", "tokenFrom": "MyToken1", "amountFrom":
"0.001", "to": "MyAddress", "tokenTo": "Token2", "maxPrice": "0.01"}' '[{"txid":
"id", "vout": 0}]'
```

e.g. Swapping 1 ETH to DFI, the command would be

```
# using symbols
defi-cli poolswap '{"from": "tnLRVU32vCfGD6...", "tokenFrom": "ETH",
"amountFrom": "1", "to": "tnLRVU32vCfGD6...", "tokenTo": "DFI", "maxPrice":
"0.01"}'
# using token ID
defi-cli poolswap '{"from": "tnLRVU32vCfGD6...", "tokenFrom": "1", "amountFrom":
"1", "to": "tnLRVU32vCfGD6...", "tokenTo": "0", "maxPrice": "0.01"}'
```

Composite swaps

Composite swaps have the same RPC structure as single swaps.

```
defi-cli compositeswap
{"from":"str","tokenFrom":"str","amountFrom":n,"to":"str","tokenTo":"str","maxPri
( [{"txid":"hex","vout":n},...] )
defi-cli compositeswap '{"from": "MyAddress", "tokenFrom": "MyToken1",
"amountFrom": "0.001", "to": "MyAddress", "tokenTo": "Token2", "maxPrice":
"0.01"}' '[{"txid": "id", "vout": 0}]'
```

e.g. Swapping 1 ETH to DUSD, the command would be

```
# using symbols
defi-cli poolswap '{"from": "tnLRVU32vCfGD6...", "tokenFrom": "ETH",
"amountFrom": "1", "to": "tnLRVU32vCfGD6...", "tokenTo": "DUSD", "maxPrice":
"0.0001"}' '[]'
# using token ID
defi-cli poolswap '{"from": "tnLRVU32vCfGD6...", "tokenFrom": "1", "amountFrom":
"1", "to": "tnLRVU32vCfGD6...", "tokenTo": "17", "maxPrice": "0.0001"}' '[]'
```

Adding liquidity

To enter an LP position, the `addpoolliquidity` RPC can be used. Equal value of assets must be provided when entering an LP. The RPC will reject the request if the price impact of adding liquidity is more than 3%. The arguments for the RPC are

- from: addresses and tokens to add to LP
- shareAddress: address to credit LP tokens to
- inputs: *(optional)* UTXOs to spend

```
defi-cli addpoolliquidity {"address":"str"} "shareAddress" (
[{"txid":"hex","vout":n},...] )
defi-cli addpoolliquidity '{"address1": "1.0@DFI", "address2": "1.0@ETH"}'
share_address '[]'
```

e.g. adding 0.1 ETH and 1 DFI of liquidity and credit LP tokens to address `tnLRVU32vCfGD6...`

```
# add liquidity from different addresses
defi-cli addpoolliquidity
'{"tnLRVU32vCfGD6...":"1.0@DFI","df1q8e3ce1j51m...":"0.1@ETH"}'
tnLRVU32vCfGD6... '[]'
# add liquidity from same address
defi-cli addpoolliquidity '{"tnLRVU32vCfGD6...": ["1.0@DFI", "0.1@ETH"]}'
tnLRVU32vCfGD6... '[]'
# auto select accounts to add liquidity with
defi-cli addpoolliquidity '{"*": ["1.0@DFI", "0.1@ETH"]}' tnLRVU32vCfGD6... '[]'
```

Removing liquidity

To exit an LP position, the `removepoolliquidity` RPC can be used. The RPC will redeem LP tokens for equal values of both pool assets.

The arguments are

- from: address to redeem LP from

- amount: amount of LP tokens to redeem
- inputs: *(optional)* UTXOs to spend

```
defi-cli removepoolliquidity "from" "amount" ( [{"txid":"hex","vout":n},...] )
defi-cli removepoolliquidity from_address 1.0@LpSymbol
```

e.g. redeeming 1 ETH-DFI LP token from address `tnLRVU32vCfGD6...`

```
defi-cli removepoolliquidity tnLRVU32vCfGD6... 1.0@ETH-DFI
```

Run a Local Node

> 🖊 **Note:**
>
> Floppynet is an ALPHA stage test network, which means it may be subject to changes. The network may experience execution failures and data persistence issues. Rollbacks are likely to happen periodically.**

---

1. Get the source code

The main development regarding MetaChain is being merged into the branch `feature/evm` of the official DeFiChain node repository. Beware that this branch will be eventually merged into the master branch and this is only a temporal development work in progress branch. For the ease of this guide we will clone the branch directly:

```
git clone -b feature/evm https://github.com/defich/ain
cd ain
```

2. Compile the node

> 🖊 **Note:**
>
> This build process is currently operational on Linux. We are actively working on the build processes for MacOS and Windows. Once complete, they should exhibit the same behavior as the Linux version.

Considering this development introduces additional dependencies, notably Rust, it is strongly suggested to utilize the make.sh script for the efficient compilation of the node. This helper script will take care of compiling and linking all of the dependencies of the DeFiChain node.

Before running the script make sure you have docker installed and running. You can follow the official guide to install, configure and run docker in yoour machine.

```
./make.sh docker_release
```

This command will compile the node dependencies and the node itself inside a docker container. Beware that this will take some time in the first run as it will download all the source code of the libraries needed and the docker images.

3. Running the node

Once the build succeeds you will find the binaries in `./buld/defichain-latest/bin/`. To run the node and connect it to floppynet run the following command:

```
./build/defichain-latest/bin/defid -devnet -connect=35.187.53.161 --daemon
```

With the `-connect=35.187.53.161` your local node will connect to the public node which is maintained by the core developers. This way it will start synchronising and finding new nodes running in Floppynet.

Note that if you are not using a snapshot the synchronisation will take some time.

An easy way to see the progress of the syncing is to run:

```
./build/defichain-latest/bin/defi-cli -devnet getblockchaininfo | head
```

This will return something like:


```
{
  "chain": "devnet",
  "blocks": 1590154,
  "headers": 1590154,
  "bestblockhash":
"0fcfca2fc40ce21ce8051b1c4cfca0ecba24d44a7b9edc4338596e5cca020685",
  "difficulty": 10057949.89351971,
  "mediantime": 1684309761,
  "verificationprogress": 1,
  "initialblockdownload": false,
  "chainwork":
"00000000000000000000000000000000000000000000007f7505aaaf5fa8b5ad7b",
```


If `blocks` and `headers` are equal and `verificationprogress` is 1 then your node is synced.

Node Logs

To check the logs go to the data folder:

- Windows: C:\Users\%username%\AppData\Roaming\DeFi Blockchain
- Mac: /Users/%username%/Library/Application Support/DeFi
- Linux: /home/%username%/.defi

Open the `debug.log` inside the `devnet/` folder with your preferred text editor

Rust debug logs

To display the Rust debug logs first stop the node:

```
./build/defichain-latest/bin/defi-cli -devnet stop
```

Then start the node with `RUST_LOG=debug`:

```
RUST_LOG=debug ./build/defichain-latest/bin/defid -devnet --daemon
```

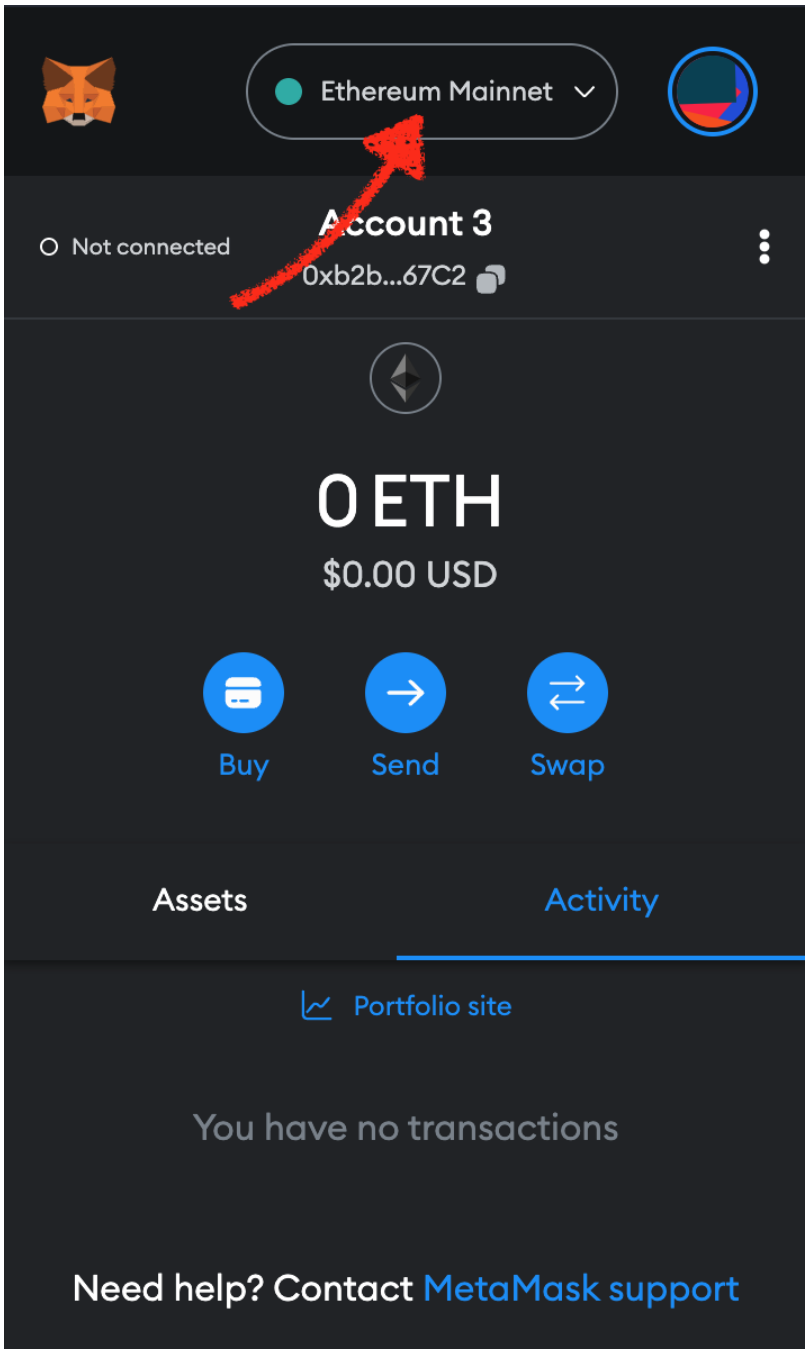
4. Connect Metamask to Floppynet

Connect Metamask to Floppynet

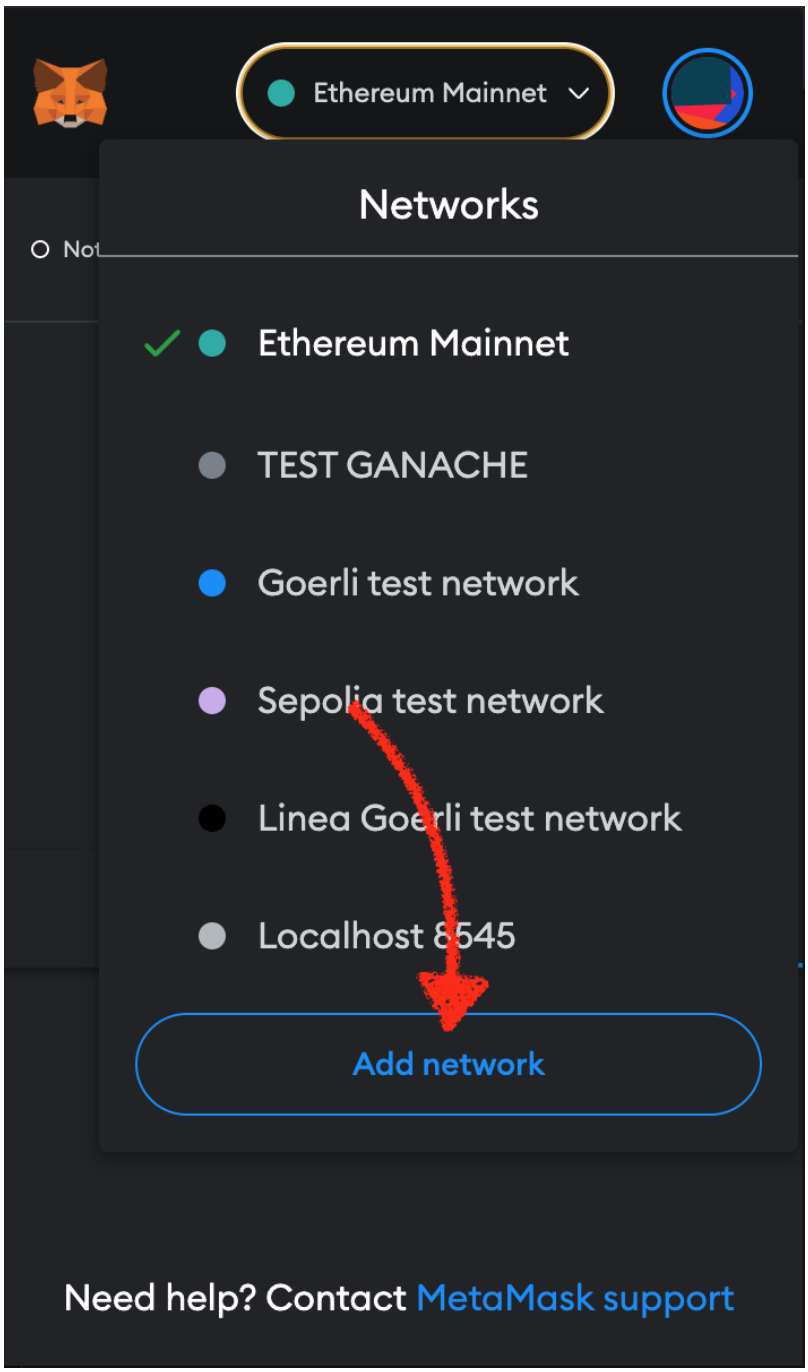
🖊 **Note:** FloppyNet is an ALPHA stage test network, which means it may be subject to changes. The network may experience execution failures and data persistence issues. Rollbacks are likely to happen periodically.**

To connect to MetaChain follow this four steps:

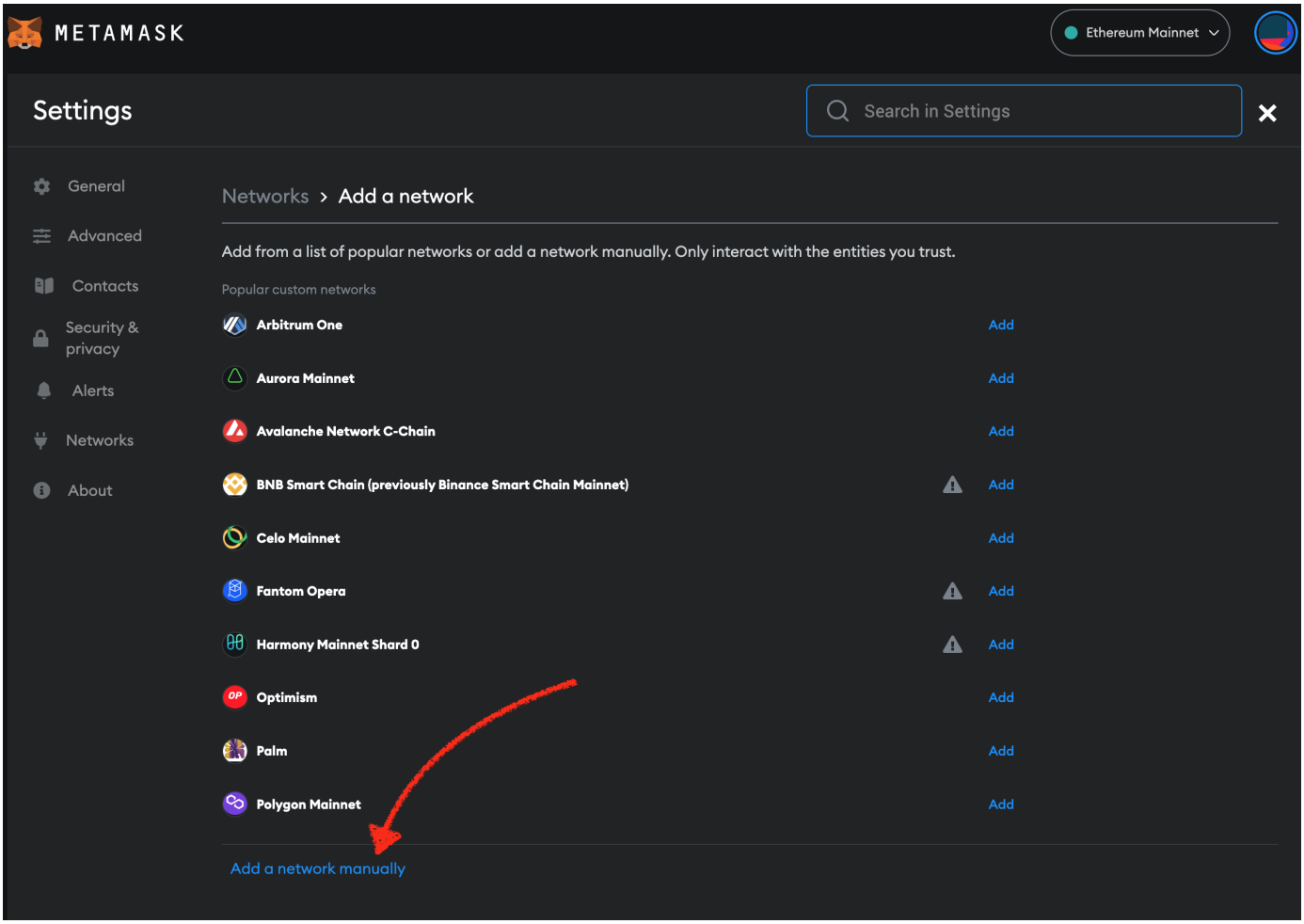
1. Open network dropdown.



2. Add network



3. Add network manually



4. Fill network form with connection data

Connection data:

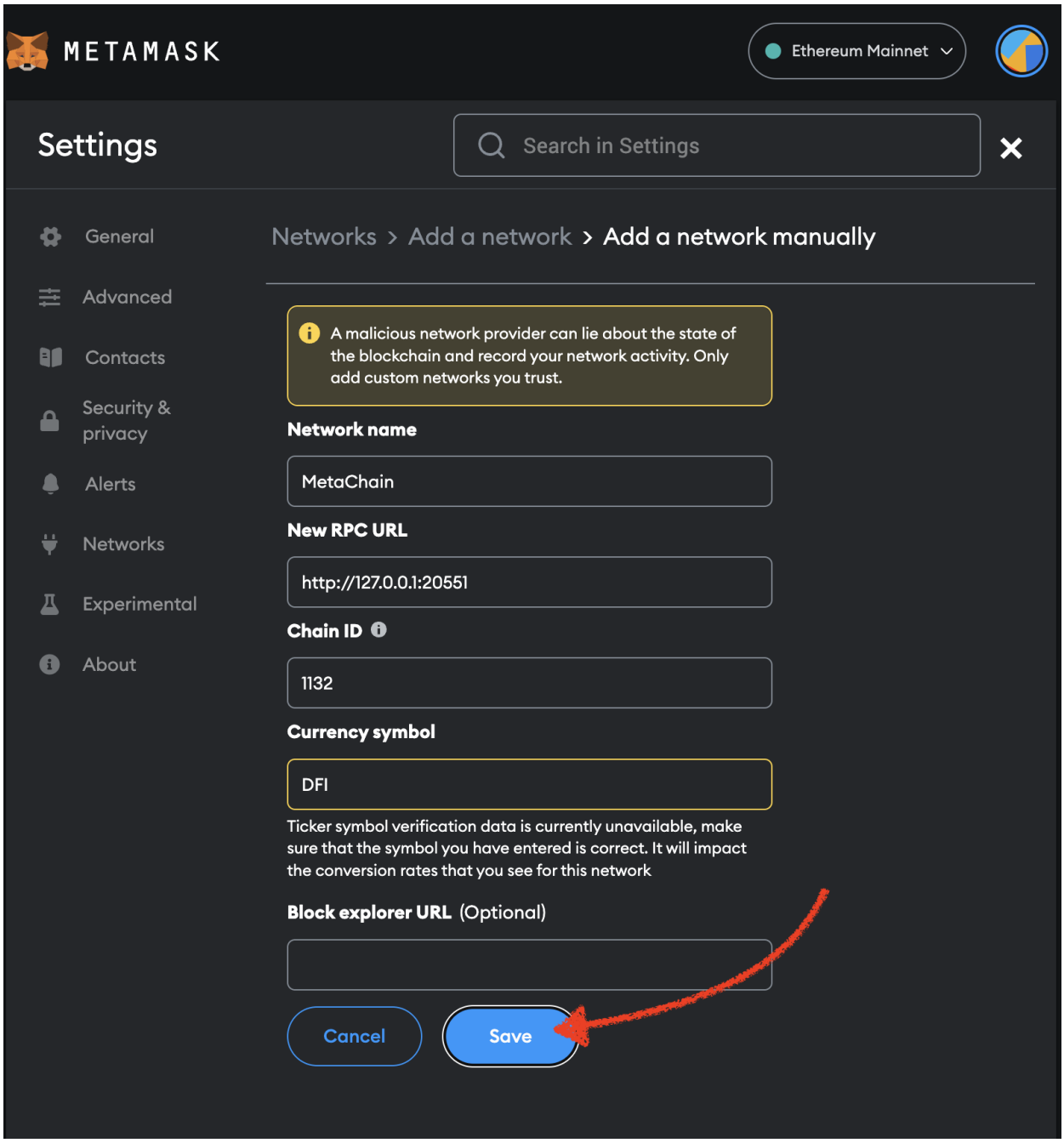
- Network name: **MetaChain** (or whatever name you prefer)

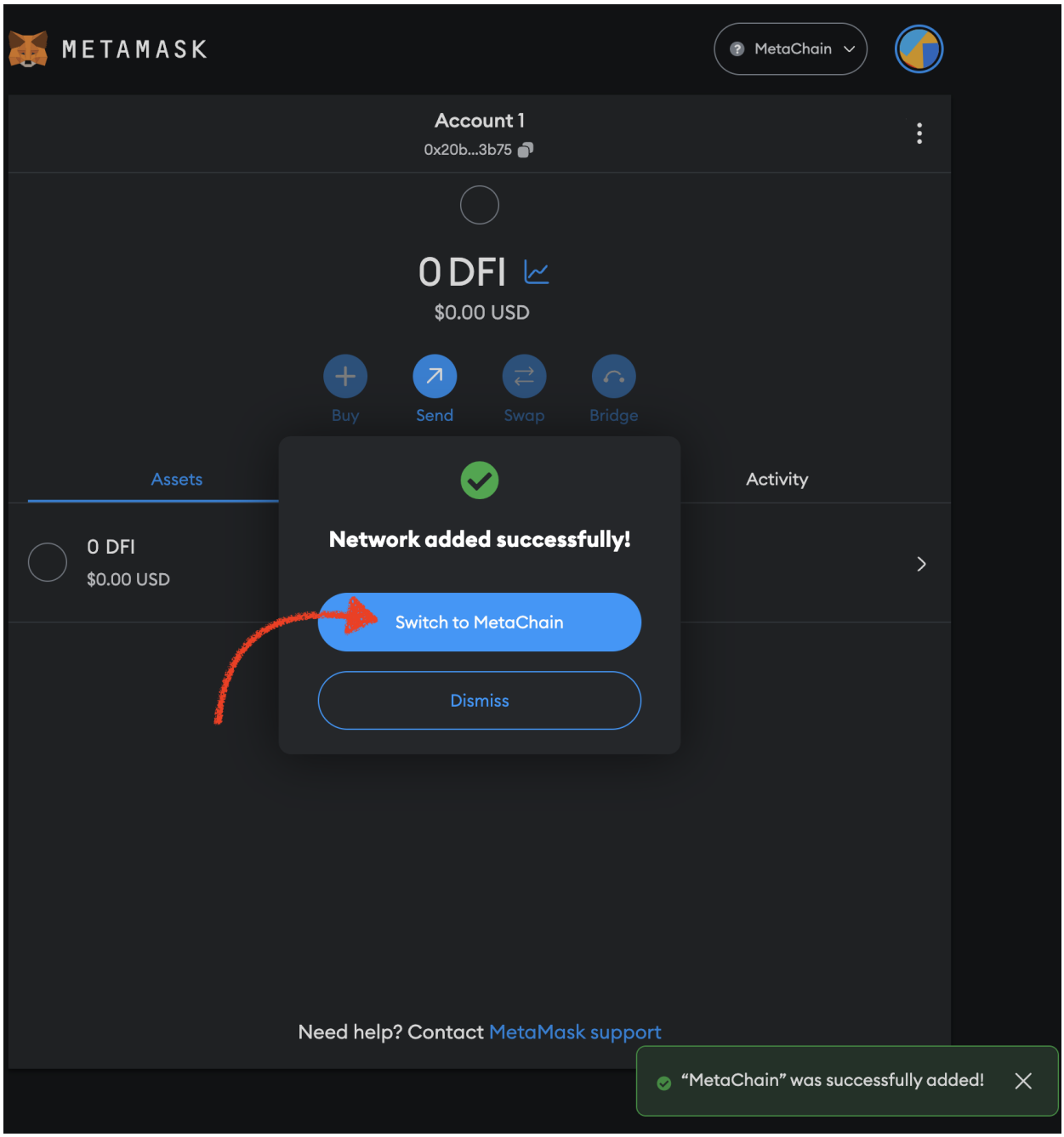
> ⚠️ **Warning: Due to potential network overloads, it is advised to use a local node over the public node to connect with Metamask. Follow the guide: Run a Local Node in Floppynet.**

- Local node (Recomended):
  - New RPC URL: **http://127.0.0.1:20551** or **http://localhost:20551**
- Public node:
  - New RPC URL: **http://35.187.53.161:20551**

> 🖊 **Note:** The public node is temporarily hosted by the core team for the easy access and convenience of early testers, and it is subject to change.

- Chain ID: **1132** (Specific to Floppynet)
- Currency symbol: **DFI**
- Block explorer URL: leave blank for now


METAMASK
Ethereum Mainnet
Settings
Search in Settings
General
Advanced
Contacts
Security & privacy
Alerts
Networks
Experimental
About
Networks > Add a network > Add a network manually
A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.
Network name
MetaChain
New RPC URL
http://127.0.0.1:20551
Chain ID
1132
Currency symbol
DFI
Ticker symbol verification data is currently unavailable, make sure that the symbol you have entered is correct. It will impact the conversion rates that you see for this network
Block explorer URL (Optional)
Cancel
Save



That's it!

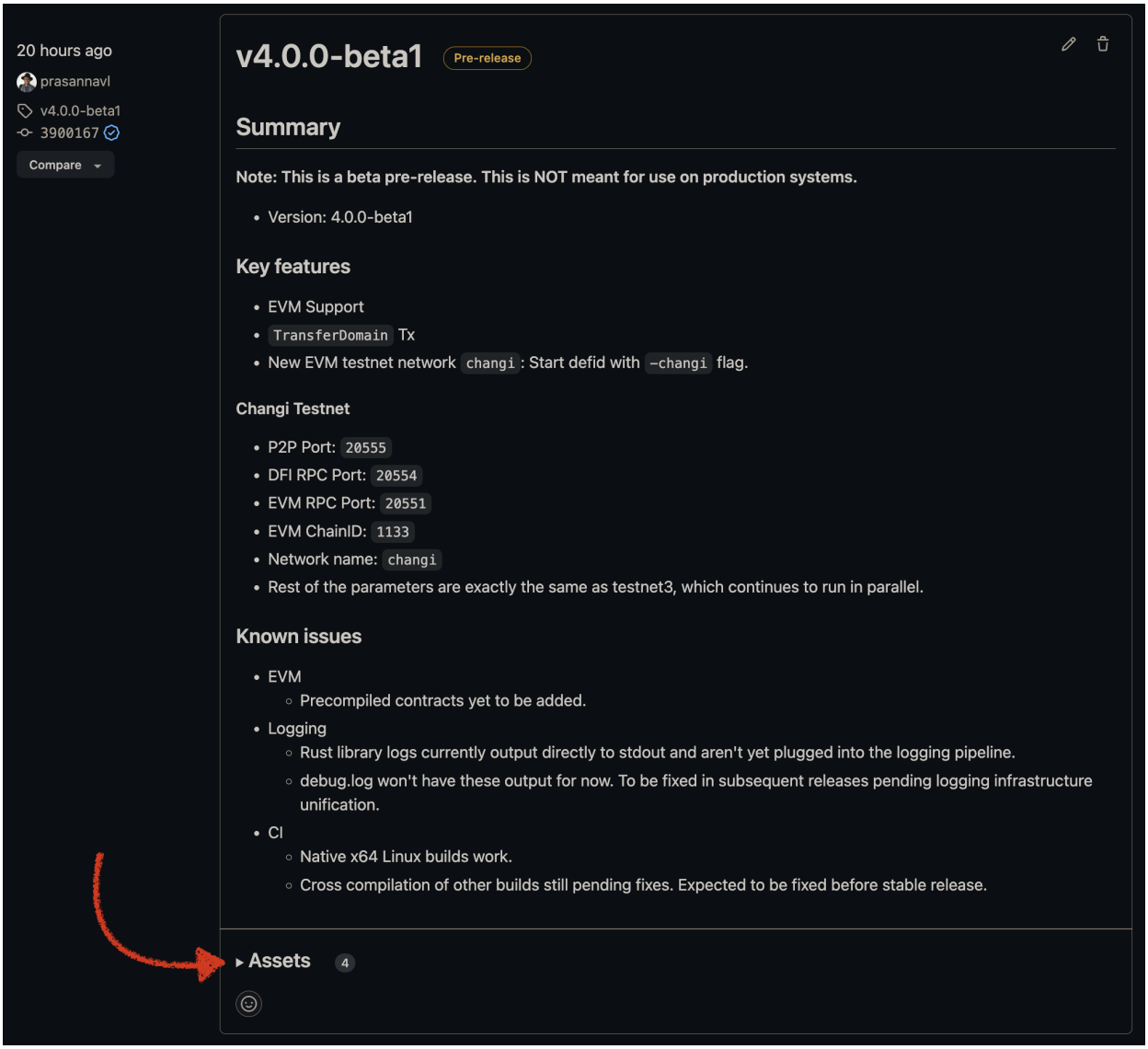
Now you can switch to the MetaChain network and see your new account, transfer funds and call and deploy smart contracts. To get some DFI, head to DeFiChain's Slack, join the `#testnet-faucet` channel and drop a message with your address, amount and reason like so:

```
Address: 0x20b50961f7ce10F70874f358d54343cB388D3b71
Amount: 50DFI
Reason: Test Smart Contracts NFT Marketplace
```

Run a Local Node in Changi

Get the binaries

Go to the official DeFiChain release section and look for the latest version of the node.



Then click on the Assets dropdown at the end of each release section. Then select the binaries for your Operating System.

Once you have downloaded the binaries extract them to your desired directory. You should see three files inside the bin folder:

- `defid`: DeFiChain node
- `defi-cli`: Command line interface to interact with the node
- `defi-tx`: utility tool to build and sign transactions on DeFiChain

For the purpose of this guide we will only be using defid and defi-cli.

Runing the node

To run the node just run the `defid` binary with the `-changi` flag

```
defid -changi
```

> If you want `defid` to run as a deamon (in the background) add the `-daemon` flag to the command above. Also if you want to have additional debug information about the EVM add `RUST_LOG=debug` to the begining of the command.
>
> `RUST_LOG=debug defid -changi --daemon`

Once the node is running it should start syncing. The process of syncing can take up to a couple of hours depending on your machine so just be patient.

To check if the node is synced you can open a new terminal, run the following command and check if the fields `blocks` and `headers` are equal:

```
defi-cli getblockchaininfo | head
```

```
> ./defi-cli -changi getblockchaininfo | head
{
  "chain": "changi",
  "blocks": 1707229,
  "headers": 1707229,
  "bestblockhash": "f938abe3dd484f5cc545d4e3ff7b2722e8841eb82cf5cb49b37742a4299b09fe",
  "difficulty": 25890570.33838039,
  "mediantime": 1686851358,
  "verificationprogress": 1,
  "initialblockdownload": false,
  "chainwork": "0000000000000000000000000000000000000000000099fa236aac0d731ac94e",
```

Connect Metamask.

To connect to MetaMask the process is the same as the one described in Connect Metamask to Floppynet and in point 4 use the Changi Testnet connection data below. For now only the

Chain ID and the optional block explorer in the Changi Testnet are different:

Network	RPC URL	Chain ID	Block Explorer
Changi Testnet	http://changi.dfi.team	1133	https://meta.defiscan.live

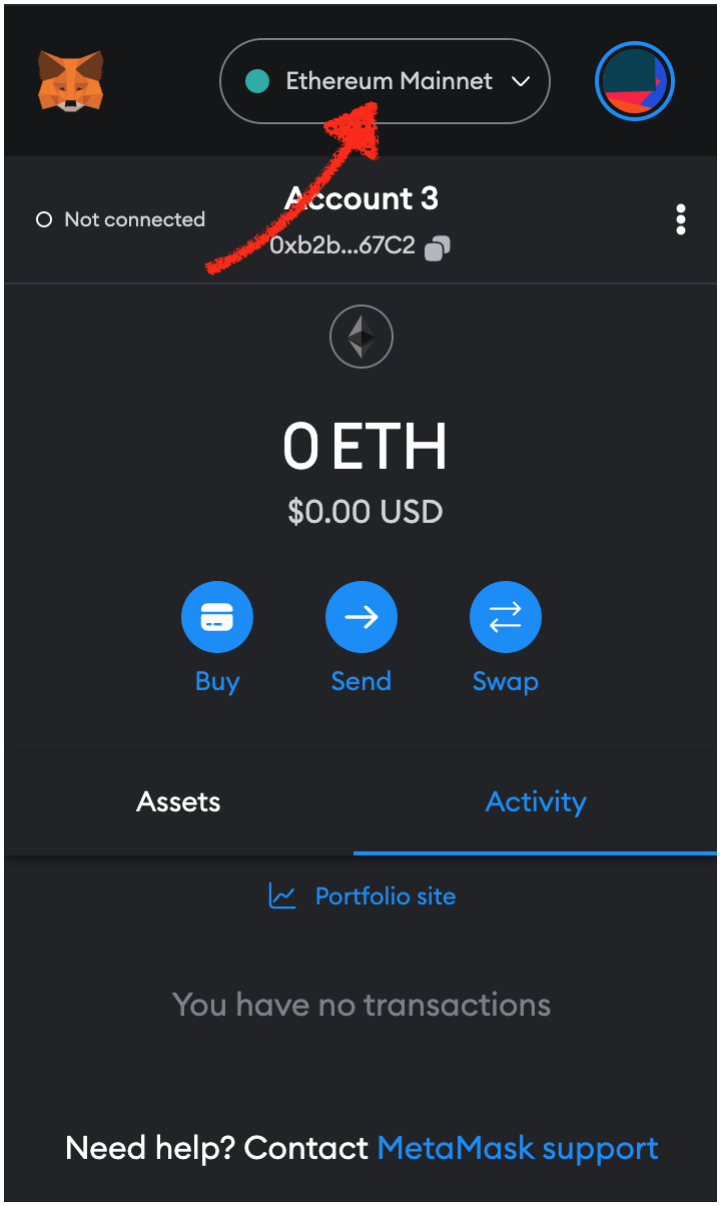
Faucet

To get Changi Testnet DFI you can use the faucet provided by the community at http://tc04.mydefichain.com/faucet/

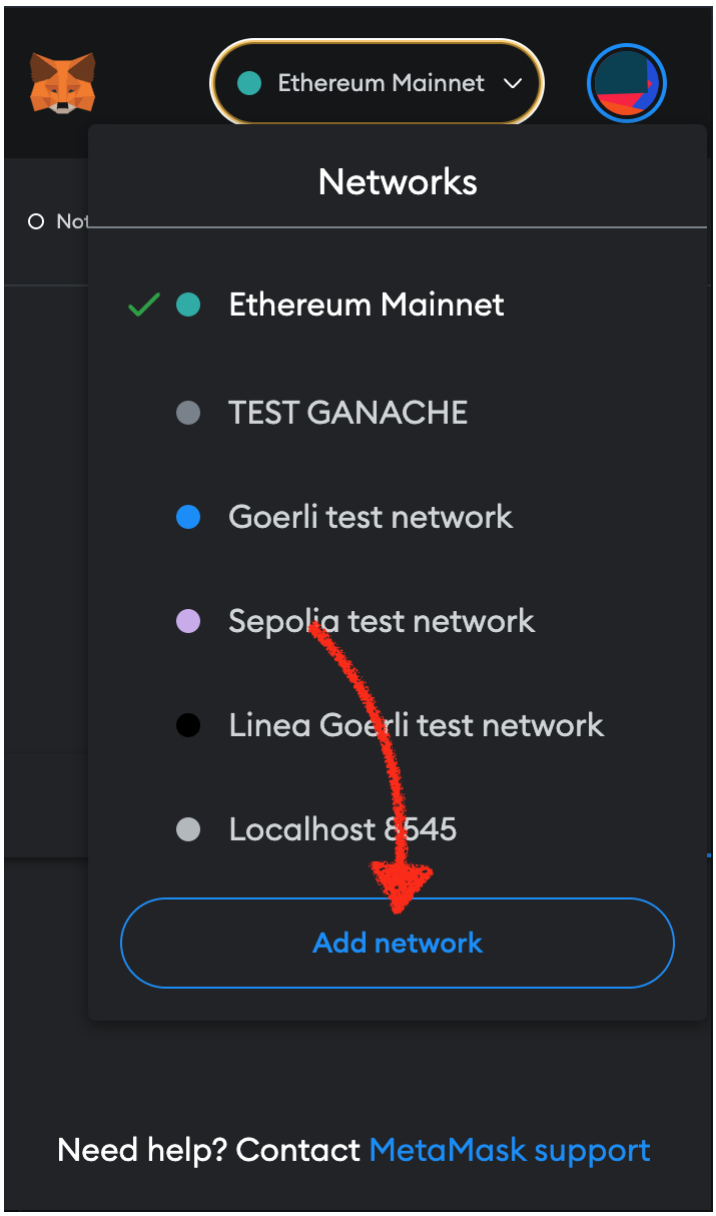
Connect Metamask to Changi TestNet

To connect to MetaChain follow this four steps:

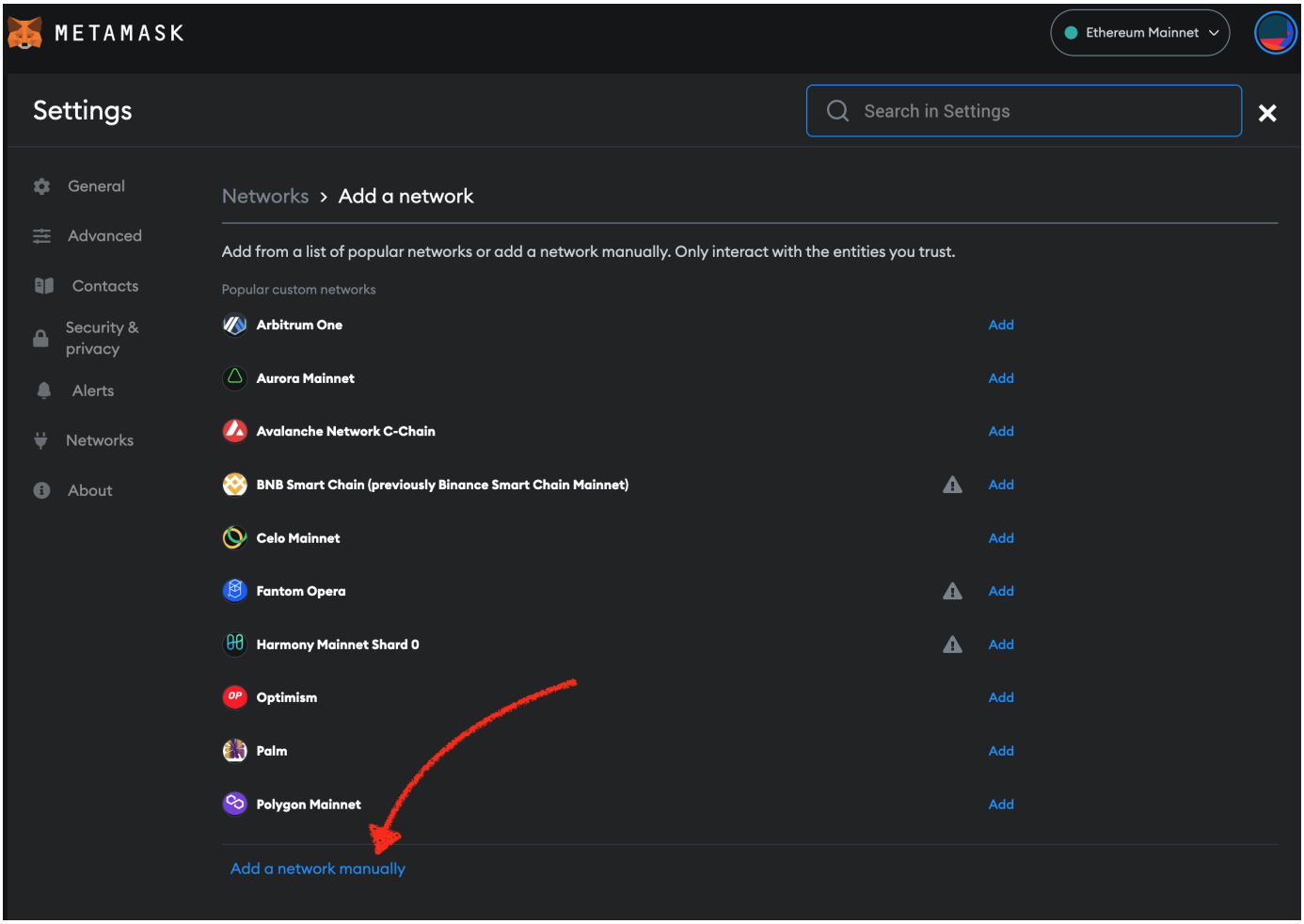
1. Open network dropdown.



2. Add network



3. Add network manually



4. Fill network form with connection data

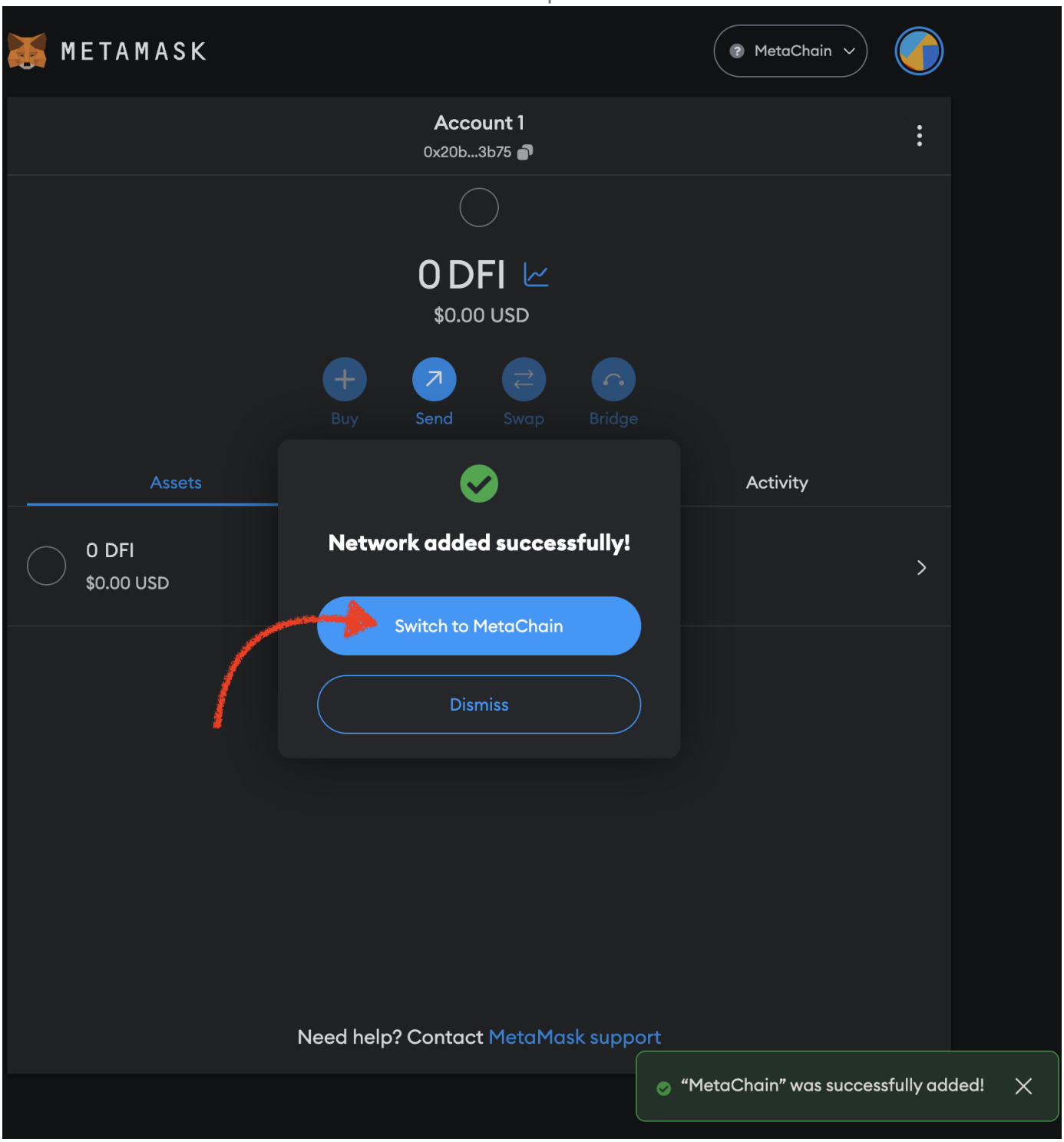
Connection data:

- Network name: **MetaChain**

> ⚠ **Warning: Due to potential network overloads, it is advised to use a local node over the public node to connect with Metamask. Follow the guide: Run a Local Node in Changi.**

- RPC URL: https://changi.dfi.team
- Chain ID: **1133** (Specific to Changi TestNet)
- Currency symbol: **DFI**
- Block explorer URL: https://meta.defiscan.live

step4



That's it!

Now you can switch to the MetaChain network and see your new account, transfer funds and call and deploy smart contracts. To get some DFI, head to DeFiChain's Slack, join the `#testnet-faucet` channel and drop a message with your address, amount and reason like so:

```
Address: 0x20b50961f7ce10F70874f358d54343cB388D3b71
Amount: 50DFI
Reason: Test Smart Contracts NFT Marketplace
```

Concepts

Hard Forks

This document lists all the hard forks in DeFiChain history and their changes.

Ang Mo Kio (v1.2.0)

Block activated: 356500

Features

- Support for DeFi Standard Token (DST), DeFi Custom Token (DCT) and DeFi Asset Token (DAT)
- The ability to create, mint and distribute tokens has opened to all users.
- Bitcoin anchor rewards in block reward
- Reserve 25% of block reward for incentive funding

Bayfront (v1.3.0)

Block activated: 405,000

Features

- DeX and liquidity pools
    - Ability to swap between tokens
    - Earn returns by providing liquidity through trading fees and liquidity incentives

Bayfront Gardens (v1.3.8)

Block activated: 488,300

- Fixed issue with rewards distribution where pool shares of less than 0.01% weren't receiving rewards
- Improved poolswap performance
- Added new RPC calls to improve wallet usability

Clarke Quay (v1.4.0)

Block activated: 595,738

Features

- Add custom rewards to pool pairs (dfip#3)
- Fix auto selecting balances
- Do not stake while downloading blocks

Dakota (v1.5.0)

Block activated: 678,000

Features

- Reduce masternode collateral requirement from 100,000 DFI to 20,000 DFI
- Full refactor of the anchoring system, for more stability and robustness
- Fix swap max price calculation. Fixes an issue where slippage protection was not activated correctly during times where the pool liquidity was low
- Fix an issue where certain transactions were occasionally stalling the blockchain
- Increase performance while staking

Dakota Crescent (v1.6.0)

Block activated: 733,000

Features

- Consensus efficiency improvements

Eunos (v1.7.0)

Block activated: 894,000

Features

- Interchain Exchange
- Update block reward scheme (dfip#8)
    - Introduce incentives for atomic swap, futures and options
    - Reduce block emmisions by 1.658% every 32,690 blocks
- Burn foundation funds (dfip#7)
- Price oracles
- Track amount of funds burnt
- Performance improvements in calculating block reward
- Increase difficulty adjustment period from 10 blocks to 1008 blocks
- Introduce 1008 block delay to masternode registration, 2016 block delay to resignation

Eunos Paya (v1.8.0)

Block activated: 1,072,000

Features

- Introduce 5 and 10 year lock for masternodes for increased staking power
- `listaccounthistory` shows pool reward types as block reward or swap fees (commission)
- Rename `getmintinginfo` to `getmininginfo`
- Add pagination to `listoracles`, `listlatestrawprices` and `listprices` RPCs
- Clear anchor and SPV dasebase when SPV database version changes
- Add support for subnode staking to make locked masternode rewards consistent
    - Normal masternodes mines using 2 subnodes.
    - 5-year lockup masternode mines using 3 subnodes.
    - 10-year lockup masternode mines using 4 subnodes.

Fort Canning (v2.0.0)

Block activated: 1,367,000

Features

- Decentralised loans
    - Collateral vaults
    - Taking loans and repayments

    - Liquidations and Auctions
- Composite swaps, swap across up to 3 pairs
- `spv_refundhtlcall` RPC gets all HTLC contracts stored in wallet and creates refunds transactions for all that have expired
- Anchoring cost has now been reduced to BTC dust (minimum possible)
- Transaction fees which were being burnt after Eunos upgrade will now be included in block rewards again

Fort Canning Road (v2.7.0)

Block activated: 1,786,000

Features

- Futures contracts
- Allow loans in a vault to be payed back with any other token through swaps to DFI
- Allow DeX fees to be applied per pool and per token (affects all pools token is in)

Fort Canning Crunch (v2.8.0)

Block activated: 1,936,000

Features

- Support for token splits

Governance Variables

Governance variables are used to track key stats and set key parameters for key blockchain utilities.

Name	Explanation
LP_LOAN_TOKEN_SPLITS*	sets block reward distribution for dToken liquidity providers
LP_SPLITS*	sets block reward distribution for crypto liquidity providers
ICX_TAKERFEE_PER_BTC	defines taker fee rate for ICX
LP_DAILY_LOAN_TOKEN_REWARD	total amount of DFI paid to loaned token holders per day
LP_DAILY_DFI_REWARD	total amount of DFI paid to liquidity providers per day
LOAN_LIQUIDATION_PENALTY	liquidation penalty percentage (as percentage)
ORACLE_BLOCK_INTERVAL	oracle update rate
ORACLE_DEVIATION	maximum permissible deviation between reported oracle values (as percentage)

**split between dToken and crypto LPs is defined in the coinbase reward scheme.*

Loans

Variable path is `ATTRIBUTES/v0/params/dfip2203/{attribute name}`.

Name	Explanation
block_period	futures contract expiration time
reward_pct	premium/discount on expiry based on price difference between DEX price and oracle price (as percentage) (see dfip#2203-A)
active	are futures enabled

Economy stats

Variable path is `ATTRIBUTES/v0/live/economy/{attribute name}`.

Name	Explanation
dfip2203_minted	amount of tokens minted in futures
dfip2203_burned	amount of collateral burned in futures
dfip2203_current	amount of tokens minted + amount currently in active contracts
dfi_payback_tokens	amount of tokens loans are paid back in by token type (DFI and DUSD)

Token attributes

Variable path is `ATTRIBUTES/v0/token/{Token ID}/{attribute name}`.

Name	Explanation
loan_collateral_factor	collateral factor percentage for token
fixed_interval_price_id	oracle price feed for token
dex_in_fee_pct	DEX fee for swapping from
dex_out_fee_pct	DEX fee for swapping to
loan_minting_enabled	allowed to mint token for loans (boolean)
loan_minting_interest	rate interest per block on minted tokens* (usually set to 0)

**this interest is in addition to the vault scheme's interest rate*

Burns

DFI and dTokens are burnt when interacting with various dApps such as DEX and loans, as well as due to extrinsic actions such as submitting CFPs (10 DFI) and DFIPs (50 DFI).

The amount of DFI and dTokens burnt can be found using the `getburninfo` RPC. A breakdown of the various sources of token burns can be found below.

Output	Description
amount	(in DFI) the cumulative amount of transaction fees paid. This is redistributed to DeFiChain masternodes.
tokens	miscellaneous burns (such as CFP/DFIP proposal fee)
feeburn	(in DFI) DFI locked up to create masternodes (20000 DFI + 11 DFI non-refundable creation fee), tokens and vaults (1 DFI + 1 DFI refundable)
auctionburn	(in DFI) 5% auction fee + vault interest paid by auction participants
paybackburn	cumulative loan principal and interest paid back
dexfeetokens	cumulative DEX input and output fees burnt. Redistributed to liquidity providers.
dfipaybackfee	loaned DFI paid back in DFI
dfipaybacktokens	loaned DFI paid back in DUSD
paybackfees	cumulative dToken loan payback penalties
paybacktokens	cumulative dToken loan tokens paid back
emissionburn	DFI burnt due to unallocated block reward (see block reward)
dfip2203	dTokens burnt from DeFiChain Futures

Releases

Glossary

Hashing

All DeFiChain hashing implentations follow the Bitcoin hashing implementation. All usage of hashes such as `CHash256` in DeFiChain are double SHA-256 hashes (such as hashing block headers and transactions). RIPEMD-160 is used when a shorter hash is desirable (for example when creating a DeFiChain address).

All DeFiChain hashing implentations follow the Bitcoin hashing implementation. The following are the key hashing functions used across various points:

- `CHash256(x)` = `SHA256(SHA256(x))`

  Example of double-SHA-256 encoding of string "hello":

  ```
  hello
  2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 (first
  round of sha-256)
  9595c9df90075148eb06860365df33584b75bff782a510c6cd4883a419833d50 (second
  round of sha-256)
  ```

- `CHash160(x)` = `RIPEMD160(SHA256(x))`

  For bitcoin addresses (RIPEMD-160) this would give:

  ```
  hello
  2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 (first
  round is sha-256)
  b6a9c8c230722b7c748331a8b450f05566dc7d0f (with ripemd-160)
  ```

Double hashing provides protection against length extension attacks and reduces collision probability.

External Resources

- SHA-2
- RIPEMD
- Bitcoin Block Hashing Algorithm

Merkle Trees

Merkle trees are binary trees of hashes. Merkle trees in DeFiChain use a double SHA-256. Merkle trees allow us to efficiently if check a transaction is part of the block - in `O(log n)` time. If, when forming a row in the tree (other than the root of the tree), it would have an odd number of elements, the final double-hash is duplicated to ensure that the row has an even number of hashes.

First form the bottom row of the tree with the ordered double-SHA-256 hashes of the byte streams of the transactions in the block.

Then the row above it consists of half that number of hashes. Each entry is the double-SHA-256 of the 64-byte concatenation of the corresponding two hashes below it in the tree.

This procedure repeats until we reach a row consisting of just a single double-hash. This is the Merkle root of the tree.

For example, imagine a block with three transactions a, b and c. The Merkle tree is:

```
d1 = dhash(a)
d2 = dhash(b)
d3 = dhash(c)
d4 = dhash(c) # a, b, c are 3. that's an odd number, so we take the c twice

d5 = dhash(d1 concat d2)
d6 = dhash(d3 concat d4)

d7 = dhash(d5 concat d6)
```

where

```
dhash(a) = sha256(sha256(a))
```

`d7` is the Merkle root of the 3 transactions in this block.

The implementation of Merkle trees can be found at `src/consensus/merkle.cpp`.

Note: Hashes in Merkle Tree displayed in the Block Explorer are of little-endian notation. For some implementations and calculations, the bytes need to be reversed before they are hashed, and again after the hashing operation.

External References

FAQ

What is subnode staking?

Masternodes that are locked in for 5 and 10 years should increased rewards of 1.5x and 2x respectively, the subnode staking system was introduced to provide consistent returns. Since each of the subnodes has a unique masternode ID, it therefore allows for multiple hashes in the same interval. Thus, having multiple subnodes is equivalent to having a higher mining hashrate, and therefore proportionally increases chances of mining a block successfully leading to higher returns on average.

- Normal masternodes uses 2 subnodes
- 5 year lockup masternode uses 3 subnodes
- 10 year lockup masternode uses 4 subnodes

While using composite swaps, are fees applied once or multiple times?

Composite swaps will require the user to fees for every DeX pool that is utilised.

For example, swapping from dLTC to DUSD has the following routing and fees

```
LTC
 ↓ (0.2%)
DFI
 ↓ (0.2%)
DUSD
```

$$fees = 1 - ((1 \times 0.998) \times 0.998) = 0.003996$$

Net fees for the swap will be 0.3996% without accounting for slippage.

How is the number of LP tokens calculated?

The total number of liquidity tokens in a pool is

$$min(\text{liquidity of A}, \text{liquidity of B})$$

When new liquidity is added, the amount of tokens returned to the user is

$$min(liq_a, liq_b)$$

where

$$liq_a = \frac{\text{amount of A added} * \text{total liquidity}}{\text{amount of A}}$$

and

$$liq_b = \frac{\text{amount of B added} * \text{total liquidity}}{\text{amount of B}}$$

What is the current block reward scheme?

The base block reward is distributed to 6 different recipients. Recipients marked with UTXO are the only transactions present in the coinbase transaction.

Recipient	Coinbase allocation	Description
Masternode (UTXO)	33.33%	Masternode rewards. Masternode also receives additional income from transaction fees.
Community Fund (UTXO)	4.91%	Community fund controlled by the Foundation, used to finance DeFiChain initiatives. ref
Anchor	0.02%	Fund to pay the Bitcoin transaction fee for anchoring.
Liquidity Pools	25.45%	Distributed to crypto-crypto LP providers.
Loans	24.68%	Distributed to dToken-DUSD LP providers.
Options	9.88%	Distributed to options holders. Not in use currently.
Unallocated	1.73%	N/A

Issues

A list of issues with breaking changes.

Issue Number	Title	URL