Warp Finance V2 Chisel Contract

1. Overview

“Chisel” is a “contract” on top of the vault where people deposit funds and the contract is able to direct those funds to different lending pair instances.

This is basically custodial, which means that there is an address that can move funds from one lending pair to another.

1) Why need a Chisel?

Like a Vault, people deposit funds (like USDC, ETH, etc), and the Vault deposits funds into different lending pairs.

The problem we’re trying to solve is the initial sending of liquidity for new lending pairs.

2) How Chisel can solve the “initial seeding” problem.

People deposit funds (like USDC, ETH, etc) into Chisel.

And Chisel deposits these funds into different leading pairs.

People can still deposit directly into the lending pairs if they want.

3) Examples

- Users deposit funds of tokens or vault_share into Chisel.

- Chisel admin deposits funds as borrow assets into lendingPair (defaultLendingPair).

- Chisel admin can move liquidity anytime they want across pairs.

- Users can withdraw at any time they want.

- When withdraw, users receive deposit funds and rewards earned from the lending pair

- Chisel calculates users' rewards using math from V2's RewardDistributor contract.

- We also have a rebalance contract that rebalance liquidities of lendingPairs.

4) Chisel depositors will get some profits by depositing tokens.

Chisel depositors and rebalancers will get some rewards.

2. Chisel Contract

1) Overview

We will have one Chisel contract for each token - ***signal token based***

- These Chisel contracts will be managed by ChiselFactory.

- It has a constant ***baseToken*** variable that is also a borrow asset of lendingPair

- Chisel contract will derive 2 contracts - ***Rebalance***, ***RewardDistribution***

Rebalance: implement all logic for rebalance pair's liquidity

RewardDistribution: implement all logic for manage interests

- When users deposit funds into Chisel, they will receive ***ChiselReceiptToken***

2) ChiselReceiptToken

These tokens will be minted when users deposit and will be burnt when withdraw.

This is an ERC20 token with option of "isVaultShare",

For users who deposit with tokens, "isVaultShare" is false.

For users who deposit with vault_shares, "isVaultShare" is true.

3) percentInterestDepositor

Percent of lendignPair's interest that will be sent to the Depositors.

Example: If this value is 90%, then 90% of interest will be sent to the Depositors.

Other 10% will be stored in Chisel for later rewards for rebalancers.

4) StackedForRebalancer

- When withdrawn, (100% - ***percentInterestDepositor***) of interests are stacked on Chisel for future rewards to the rebalancers later.

- StackedForRebalancer is for those stacked amounts.

- This will be increased whenever calling the withdraw function.

- This will be decreased whenever calling rebalance function

3. Chisel Methods

1) Deposit

(1) depositToken

- Overview

Using this method, users will deposit tokens directly into Chisel. These Chisel funds are deposited into Vault. Later Chisel admin can move these funds into ***LendingPair***.

- Code Example

```
function depositToken(uint256 amount) external {

   // mint recept tokens to the depositors

   receiptNativeToken.mint(msg.sender, amount);
   // deposit tokens into Vault

   vault.deposit(token, msg.sender, address(this), _amount);

 }
```

(2) depositVaultShares

- Overview

If users deposit funds into Vault before, they have a ***vault_share*** amount that displays deposited balance in the Vault. To save gas, they can use this ***vault_share*** to deposit into Chisel. There is no need to deposit these tokens into the Vault again.

- Code Example

```
function depositVaultShares(uint256 vault_share) external {

   // record user deposited token balance

   receiptVaultShareToken.mint(msg.sender, vault_share);


   // transfer vault_shares from depositor to chisel

   vault.transfer(token, msg.sender, address(this), vault_share);

 }
```

(3) checkChiselLiquidity

- Overview

The ***Chisel*** admin will call this method to see how many tokens (***vault_shares***) are on the ***Chisel*** now. These funds will be deposited into defaultLendingPair later.

- Code Example

```
function checkChiselLiquidity() external {
   // total token balance can be deposited into LendingPair later
   uint256 total = vault.toShare(pair.asset(), amount, false);

   // StackedForRebalancer is genereated whenever withdraw
   // this is for futher rewards to the rebalancers
   return total - StackedForRebalancer;
 }
```

2) AddLiquidityToLendingPair

- Overview

Chisel admin deposits funds as Borrow assets into ***LendingPair***.

- Code Example

```
function addLiquidityToLendingPair(uint256 amount)

   public

   onlyAdmin

 {
   // check if has enough amount to deposit via calling checkChiselLiquidity()

   // if defaultLendingPair is set, move vault_share amounts of Chisel to lendingPair
as borrow

   // if not, it will be reverted

 }
```

3) moveLiquidity

- Overview

At any time, Chisel admin can move liquidity from one lending pair to another.

- Keys

This method can be called by only Chisel ***Admin***.

- Code Example

```
function moveLiquidity(

   IBSLendingPair from,

   IBSLendingPair to,

   uint256 amount

 ) public onlyAdmin {

   // check if `from` pair has enough balance

   // check if two LendingPair's borrow assets Chisel’s baseToken
   // move liquidity from using Vault's `transfer` function

 }
```

4) Withdraw

Before withdrawing from LendingPair, it checks if user funds are deposited into LendingPair.

- If yes, withdraw from lendingPair and return some rewards too.

- If not, withdraw from Chisel without rewards.

(1) Withdraw from lendingPair

- Chisel withdraws funds from LendingPair and returns to depositors.
- Here some rewards and interest distribution works.
- Any user can call this method, but if their deposited amount is less than the withdrawal amount, this will be reverted.

```
function withdraw(

       IBSLendingPair _pair,

       uint256 _amount,

       address _receipient,

       bool _vaultShare

   ) external afterInitialized returns (bool) {

       // 1. check if withdraw amount does not exceed than deposit funds

       // we can check this by comparing the ReceiptToken.balanceOf(msg.sender)


       // 2. calculate pair_total_withdraw_vault_share from _amount
```

```
        // 3. withdraw from lendingPair to Chisel

        // 4. calculate stacks_for_rebalance_amount
        // pair_total_withdraw_vault_share * (100 - interestDepositorPercent) / 100)

        // 5. send stacks_for_rebalance_amount from chisel to RewardDistribution

        // 6. depositor's depositr_interests_withdraw_vault_share =
pair_total_withdraw_vault_share - stacks_for_rebalance_amount


        /* 7. depositor_incentive_interests = 0
            if (!_vaultShare) {
                depositor_incentive_interests =
RewardDistribution.calcUserIncentiveInterests(_pair, msg.sender, _amount)
            }
        */

        // 8. depositr's user_total_withdraw_vault_share =
depositr_interests_withdraw_vault_share + depositor_incentive_interests

        // 9. withdraw user_total_withdraw_vault_share to _receipient

        // 10. burn receipt tokens
    }
```

(2) withdraw from Chisel

- Chisel withdraws funds from Chisel to depositors without rewards.

4. RewardDistribution

1) overview

There are 2 types of rewards in Chisel.

(1) ***PairIncome***

When depositing into lendingPair, there are some interests.
Since chisel deposit funds into lendingPair, they will receive interests called ***PairIncome***

PairIncome = PairInterests + PairIncentiveInterests

- ***PairInterests:*** interests earned from all lendingPairs
- ***PairIncentiveInterests:*** specific interests earned from some incentive pairs.

(2) ***WarpRewards***

When deposited into Chisel, users receive receiptTokens.

They can deposit those receiptTokens into MasterChef contract and it will generate WARP tokens as rewards.

This only works if users deposit receiptTokens that they received by depositing tokens, not vault_shares.

2) PairIncome

(1) This ***PairIncome*** will be divided into two parts.
- ***interestDepositorPercent %*** will be sent to the depositors.
- (100 - ***interestDepositorPercent***)% will be stored in Chisel to reward rebalancers later.

(2) This ***PairIncome*** will be collected whenever calling the ***addIncome*** method.
- Rebalancers can only call this method.
- They will receive some bonus like when calling the ***rebalance*** method.

5. Rebalancer

1) Overview

- This is a contract responsible for rebalancing and managing incentiveInterests earned from some specific incentive pairs.

- We're using wildCredit's mechanism for incentiveInterests of Rebalancer.

https://github.com/WildCredit/contracts/blob/master/contracts/VaultRebalancer.sol

https://github.com/WildCredit/contracts/blob/master/contracts/Vault.sol

2) IncentiveInterests of Rebalancer

This is a contract responsible for rebalancing and managing incentiveInterests earned from some specific incentive pairs.

3)

You said that “Periodical interest generation mechanism”

FAQ

1. How to check if user funds are deposited into LendingPair or not

2. “Accurate interests periodically and stack rewards” vs “accurate whenever withdraw”?

3. How to check if rebalance is required

In wildCredit, there’re two main features in Rebalancer.

1. Rebalance of lendingPair’s liquidity
2. Distribution of income (or interest) from lendingPairs.

This is my thought.
Like wildCredit, I don’t want to care if interest are “general interests from all lendingPairs” or “incentive interests from all some specific incentive pools”.
We can manage both of them as “income” of lendingPair.
And we will distribute this income into “depositors” and “rebalancers”.

What do you think?
In this case, shall we provide some rewards when rebalancers calling the “distribution” function of Reabalancer?

4. We need to care about 2 interests (and also masterchef)

- One is interests from all lendingPairs

This is a simple logic that divides interests into two parts
One is for depositors and the another is stacking for future rebalancer rewards.

- The second one is “incentive interests” from some “incentive lendingPairs”.