1. Token Contracts
2. TimeERC20.sol

-interface IERC20

-library LowGasSafeMath

-abstract contract ERC20 is IERC20

-library Counters

-interface IERC2612Permit

-abstract contract ERC20Permit is ERC20, IERC2612Permit

-interface IOwnable

-contract Ownable is IOwnable

-contract VaultOwned is Ownable

-contract TimeERC20Token is ERC20Permit, VaultOwned

2)wMEMO.sol

-interface IERC20

-library LowGasSafeMath

-library Address

-contract ERC20 is IERC20

-library SafeERC20

-interface IMEMO is IERC20

-contract wMEMO is ERC20

3)MemoriesERC20.sol

-library LowGasSafeMath

-library Address

-interface IERC20

-abstract contract ERC20 is IERC20

-library Counters

-interface IERC2612Permit

-abstract contract ERC20Permit is ERC20, IERC2612Permit

-contract OwnableData

-contract Ownable is OwnableData

-contract MEMOries is ERC20Permit, Ownable

1. Staking Contracts

1)Staking.sol

-library LowGasSafeMath

-interface IERC20

-library Address

-library SafeERC20

-contract OwnableData

-contract Ownable is OwnableData

-interface IMemo is IERC20

-interface IWarmup

-interface IDistributor

-contract TimeStaking is Ownable

2)StakingDistributor.sol

-library LowGasSafeMath

-interface IERC20

-library Address

-contract OwnableData

-contract Ownable is OwnableData

-interface ITreasury

-contract Distributor is Ownable

3)StakingHelper.sol

-interface IERC20

-interface IStaking

-contract StakingHelper

4)StakingWarmup.sol

-interface IERC20

-contract StakingWarmup

3.Bond Contracts

1) BondDepository.sol

-interface IOwnable

-contract OwnableData

-contract Ownable is OwnableData

-library LowGasSafeMath

-library Address

-interface IERC20

-library SafeERC20

-library FullMath

-library FixedPoint

-interface ITreasury

-interface IBondCalculator

-interface IStaking

-interface IStakingHelper

-contract TimeBondDepository is Ownable

2)EthBondDepository.sol

-interface IOwnable

-contract Ownable is IOwnable

-library LowGasSafeMath

-library Address

-interface IERC20

-library SafeERC20

-library FullMath

-library FixedPoint

-interface AggregatorV3Interface

-interface ITreasury

-interface IStaking

-interface IStakingHelper

-interface IWAVAX9 is IERC20

-contract TimeBondDepository is Ownable

3)StandardBondingCalculator.sol

-library FullMath

-library Babylonian

-library BitMath

-library FixedPoint

-library LowGasSafeMath

-interface IERC20

interface IUniswapV2ERC20

interface IUniswapV2Pair is IUniswapV2ERC20

interface IBondingCalculator

contract TimeBondingCalculator is IBondingCalculator

4.Treasury Contract

library LowGasSafeMath

library Address

contract OwnableData

contract Ownable is OwnableData

interface IERC20

library SafeERC20

interface IERC20Mintable

interface ITIMEERC20 is IERC20Mintable, IERC20

interface IBondCalculator

contract TimeTreasury is Ownable

5.WonderZapIn.sol

abstract contract Context

abstract contract Ownable is Context

interface IERC20

library Address

library SafeERC20

interface IERC20Metadata is IERC20

abstract contract ZapBaseV2\_1 is Ownable

abstract contract ZapInBaseV3\_1 is ZapBaseV2\_1

library Babylonian

interface IWETH

interface IUniswapV2Factory

interface IUniswapV2Router02

interface IUniswapV2Pair

interface ITimeBondDepository

contract Wonderland\_ZapIn\_V1 is ZapInBaseV3\_1

-about BCV value

BCV is Bond control value.

It is controlled by adjust() function of TimeBondDepository contract in BondDepository.sol file.

    function adjust() internal {

        uint timeCanAdjust = adjustment.lastTime.add32( adjustment.buffer );

        if( adjustment.rate != 0 && block.timestamp >= timeCanAdjust ) {

            uint initial = terms.controlVariable;

            uint bcv = initial;

            if ( adjustment.add ) {

                bcv = bcv.add(adjustment.rate);

                if ( bcv >= adjustment.target ) {

                    adjustment.rate = 0;

                    bcv = adjustment.target;

                }

            } else {

                bcv = bcv.sub(adjustment.rate);

                if ( bcv <= adjustment.target ) {

                    adjustment.rate = 0;

                    bcv = adjustment.target;

                }

            }

            terms.controlVariable = bcv;

            adjustment.lastTime = uint32(block.timestamp);

            emit ControlVariableAdjustment( initial, bcv, adjustment.rate, adjustment.add );

        }

    }

-How to manage APY for staking

**APY** stands for annual percentage yield. It is contolled in Distributor contract in StakingDistributor.sol file.

-About Treasury

The contract of TimeTreasury in Treasury.sol file is explained about Treasury.

-About rebase

A rebase() function is defined in TimeStaking contract of staking.sol.

    function rebase() public {

        if( epoch.endTime <= uint32(block.timestamp) ) {

            Memories.rebase( epoch.distribute, epoch.number );

            epoch.endTime = epoch.endTime.add32( epoch.length );

            epoch.number++;

            if ( address(distributor) != address(0) ) {

                distributor.distribute();

            }

            uint balance = contractBalance();

            uint staked = Memories.circulatingSupply();

            if( balance <= staked ) {

                epoch.distribute = 0;

            } else {

                epoch.distribute = balance.sub( staked );

            }

            emit LogRebase(epoch.distribute);

        }

    }

Equations

-minting

Debt ratio:

It is defined debtRatio() function in TimeBondDepository contract in BondDepository.sol file.

    function debtRatio() public view returns ( uint debtRatio\_ ) {

        uint supply = Time.totalSupply();

        debtRatio\_ = FixedPoint.fraction(

            currentDebt().mul( 1e9 ),

            supply

        ).decode112with18() / 1e18;

    }

All equations in minting part are controlled in TimeBondDepository contract.

bondprice

premium

debtRatio

bondPayout...

-Time Supply

Equations related to Time supply is controlled in Distributor contract of StakingDistributor.sol file.

-deposit

This function determines deposite bond.

    function deposit(

        uint \_amount,

        uint \_maxPrice,

        address \_depositor

    ) external returns ( uint ) {

        require( \_depositor != address(0), "Invalid address" );

        require(msg.sender == \_depositor || allowedZappers[msg.sender], "LFNA");

        decayDebt();

        uint priceInUSD = bondPriceInUSD(); // Stored in bond info

        uint nativePrice = \_bondPrice();

        require( \_maxPrice >= nativePrice, "Slippage limit: more than max price" ); // slippage protection

        uint value = treasury.valueOf( address(principle), \_amount );

        uint payout = payoutFor( value ); // payout to bonder is computed

        require( totalDebt.add(value) <= terms.maxDebt, "Max capacity reached" );

        require( payout >= 10000000, "Bond too small" ); // must be > 0.01 Time ( underflow protection )

        require( payout <= maxPayout(), "Bond too large"); // size protection because there is no slippage

        // profits are calculated

        uint fee = payout.mul( terms.fee )/ 10000 ;

        uint profit = value.sub( payout ).sub( fee );

        uint balanceBefore = Time.balanceOf(address(this));

        /\*\*

            principle is transferred in

            approved and

            deposited into the treasury, returning (\_amount - profit) Time

         \*/

        principle.safeTransferFrom( msg.sender, address(this), \_amount );

        principle.approve( address( treasury ), \_amount );

        treasury.deposit( \_amount, address(principle), profit );

        if ( fee != 0 ) { // fee is transferred to dao

            Time.safeTransfer( DAO, fee );

        }

        require(balanceBefore.add(profit) == Time.balanceOf(address(this)), "Not enough Time to cover profit");

        // total debt is increased

        totalDebt = totalDebt.add( value );

        // depositor info is stored

        bondInfo[ \_depositor ] = Bond({

            payout: bondInfo[ \_depositor ].payout.add( payout ),

            vesting: terms.vestingTerm,

            lastTime: uint32(block.timestamp),

            pricePaid: priceInUSD

        });

        // indexed events are emitted

        emit BondCreated( \_amount, payout, block.timestamp.add( terms.vestingTerm ), priceInUSD );

        emit BondPriceChanged( bondPriceInUSD(), \_bondPrice(), debtRatio() );

        adjust(); // control variable is adjusted

        return payout;

    }

The other variables are defined when it is deployed contract to network.