Jeffrey Scholz Jan 7 4 min read

cUSDC V3 (Compound V3) as a non-standard Rebasing Token, CometExt.sol

The Compound V3 contract behaves like a rebasing ERC 20 token. A rebasing token is a token which has an algorithmically adjusted supply rather than a fixed one. The “token” here represents the present value of positive USDC balances. That is, lenders can transfer the present value of their principal to other addresses as if it were an ERC 20 token. Since the principal value is generally increasing due to interest accrual, this ERC 20 token is rebasing upwards over time.

Compound V3 does not use a token vault standard (e.g. ERC 4626) to track “shares” of the lending pool.

As we noted in our discussion of principal and present value, a user may have deposited 100 USDC but have a credit of 110 USDC due to interest accrued — the 110 is the present value. It is this unit of account that the ERC20 functionality of Compound V3 manages.

Prerequisites

Users must be familiar with interest indexes and Compound V3’s notion of present value and principal value. The terms Compound V3 and Comet are used interchangeably in this article since Comet is the name of the primary smart contract which has the functions we discuss here.

Structure of this article

Each heading will discuss an ERC 20 function that Compound V3 implements, and how it implements the function. Some of the functions are not part of the ERC 20 standard but are relevant to this discussion.

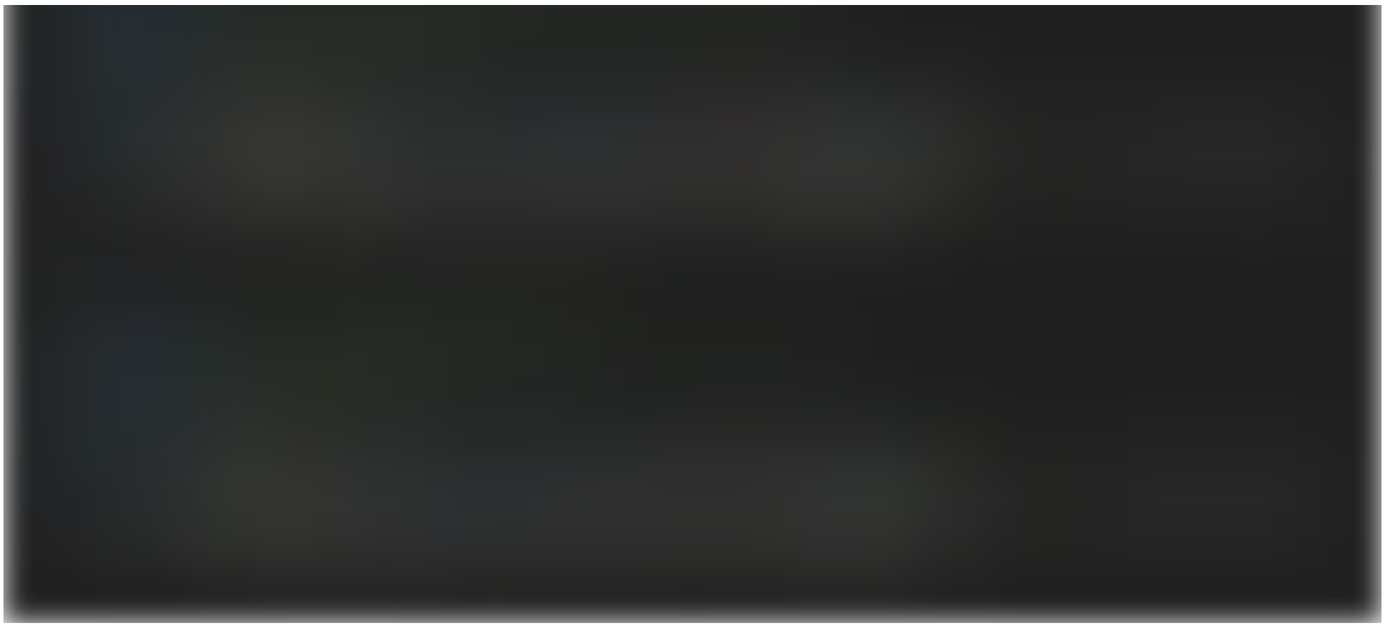
totalSupply and totalBorrow (Comet.sol)

The totalBorrow, as the name implies, is the total amount of USDC borrowed. That is, it is the present value of the debt. We can corroborate that with what the contract returns and what we see on the Compound platform. This is not just the amount of USDC borrowers withdrew from the platform — it includes the interest accrued on the borrowed USDC.

Below we show a screenshot of the Compound V3 UI showing this value and Etherscan returning the result of the totalBorrow() function.

Similarly, the totalSupply() is not the amount of USDC lenders deposited into Compound — it is the *present value* of the total deposits.

The totalSupply and totalBorrow code screenshotted below should the relationship to the present value clear.



Below we have screenshotted two queries of totalSupply. Note that the totalSupply has increased in the second screenshot on the right.

The totalSupply() function behaves as totalSupply() from ERC 20.

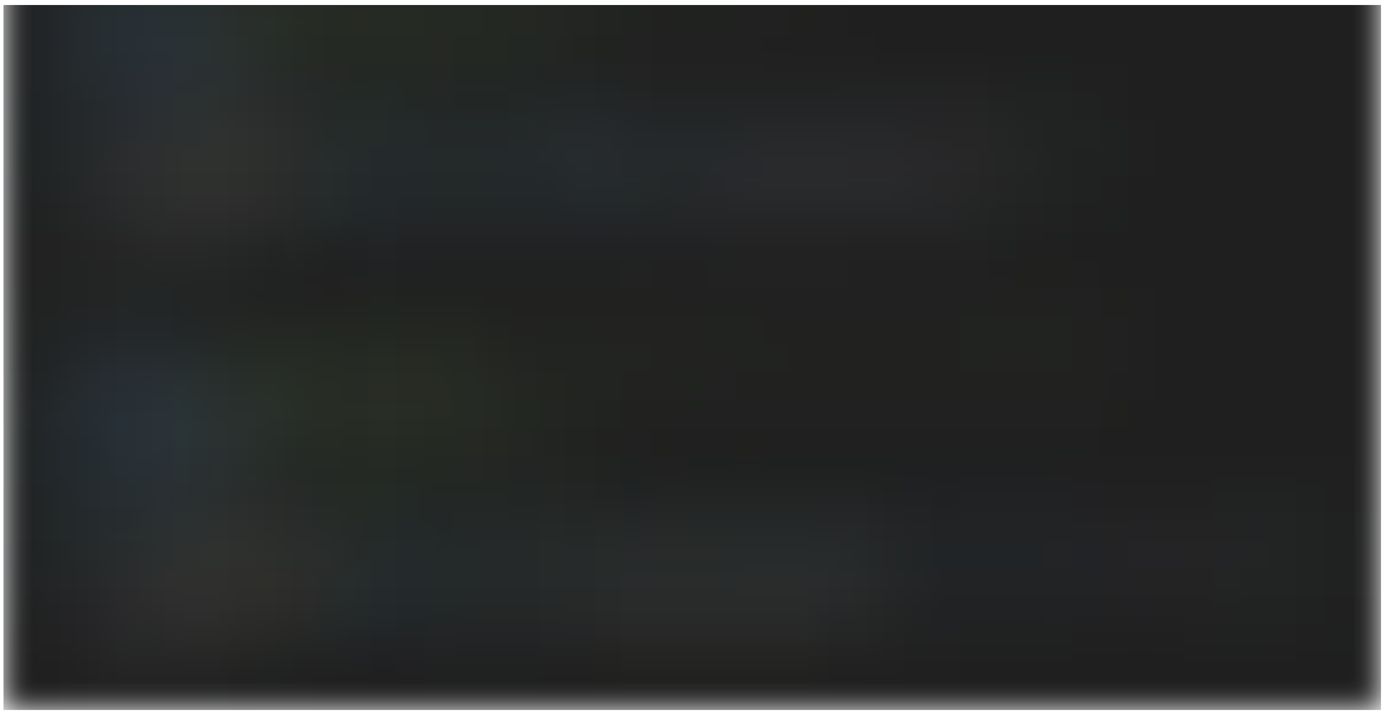
Borrowers are not able to transfer debt, so totalBorrow is not used for any token-like interfaces.

balanceOf (Comet.sol)

BalanceOf was already covered in our discussion about principal and present value, so we won't repeat it here.

transfer and transferFrom (Comet.sol)

Both transfer and transferFrom will transfer the present value of the lender. The amount parameter is measured in present value.



Both these functions call transferInternal under the hood. The right way to transfer the entire balance in Compound V3 is to transfer uint256 max value. Specifying the entire balance can be a bit tricky because it increases every second.

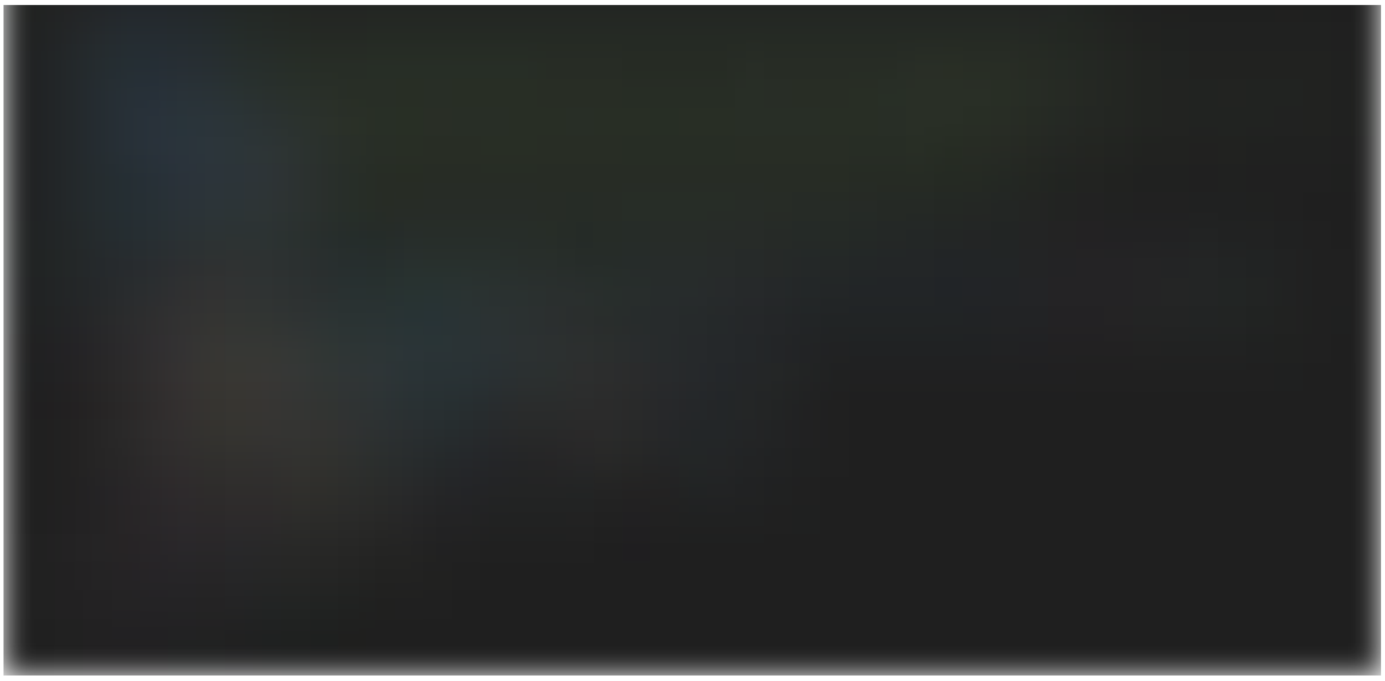
Note that there is no mechanism for borrowers to transfer collateral to other addresses, as transferCollateral is only internal. This function is used for liquidations.

The remainder of the ERC 20 functions are in CometExt.sol

Because of the 24 kb deployment limit, Comet splits off some of its functionality to CometExt.sol using the fallback extension pattern. The majority of the functions in CometExt are related to the ERC 20 functionality.

approve (CometExt.sol)

The approve() functionality for cUSDCv3 is nonstandard in that it only accepts type(uint256).max or zero. Since the balance of an account is constantly changing due to the rebasing, it isn't possible to give someone an allowance for exactly the entire balance, since the balance keeps increasing as interest accumulates.



approve() calls allowInternal under the hood which is worth examining separately

allow() and allowInternal()

The function allow() not part of ERC20, but it behaves the same as approve() in that it gives an address maximum allowance. Both approve() and allow() use allowInternal() under the hood to accomplish giving an address maximum allowance.

Because approve is essentially binary, allow behaves like approve except it takes a boolean argument to give approval for the full value instead of a uint256.

The “allowances” storage variable is kept in CometStorage.sol as isAllowed.

Allowance is all or nothing in Compound V3. That's why approve only accepts the maximum uint256 value. There is no storage variable for storing allowance as a number like traditional ERC 20 tokens do.

allowance (CometExt.sol)

Allowance is binary, you only have approval for the max uint256 value or zero. Any other value will revert.

hasPermission simply returns a Boolean value signifying an address has unlimited approvals or none at all.

allowBySig() is a non-standard ERC 20 permit() function

CometStorage exposes mapping(address => uint256) public userNonce as a public variable rather than the nonces(address owner) external returns (uint) specified by EIP 2612.

name() and symbol() (CometExt.sol)

The functions name() and symbol() are optional ERC 20 functions that return strings.

CometExt.sol does not store these values in string variables, but instead in immutable bytes32 variables for gas efficiency purposes. Solidity does not allow for casting bytes32 to strings directly, so the immutable variables are converted to strings on the fly using the code below.

One lesser known detail about bytes1, bytes2, …, and bytes32 datatypes in Solidity is that they can be indexed at the byte level like byte arrays. For example,

```
contract Example {
    bytes32 immutable x = 0x3300000000000000000000000000000000000000000000000000000000000000;

     function main() external pure returns (bytes1) {
        return x[0]; // returns 0x33
    }
}
```

CometExt initializes a bytes array in memory and copies over the characters stored in the name32 and symbol32 bytes32 immutable variables, then casts that to a string.

Recent Posts See All

Calling functions in CometExt

cast.

Subscribe to our newsletter Subscribe Now

We do not sell your information to anyone. Period.

Web3 Blockchain Bootcamp