Compound Comet Specification

Compound Comet

Overview

Given that most borrowing activity in DeFi today consists of supplying volatile crypto assets and borrowing a single borrowable base token, we aim to achieve greater capital efficiency (including gas costs) by building a specialized protocol which allows you to supply volatile assets, and borrow only a single (e.g. stable) coin.

Value Proposition

1. More capital efficiency (more dollars for same collateral*)
2. Optimized for common usage (crypto collateral, USDC** borrows)
3. Fine-grained access controls for delegating account management
4. Internalize liquidation / profits

* Collateral here exclusively refers to an ERC-20 token (or a similar token standard for a different blockchain). The native token (e.g. Ether) must be wrapped as either WETH or LIDO to be used as collateral.

** Base token refers to the single borrowable asset, e.g. USDC. We may also refer to this as the base token.

Architecture

Contracts

As the goal of Comet is to be highly optimized for a particular use case, we seek to minimize the number of contracts involved. The protocol is implemented primarily in a monolithic contract. We assume all math operations revert in the case of overflow or underflow. We assume all values are unsigned 32-byte integers unless otherwise specified.

Prices

This section is incomplete.

The protocol will use the v2 price feed

Interest Rates

Unlike Compound v2, where the supply rate is derived from the borrow rate and reserve factor, in Compound Comet the protocol has a distinct borrow and supply rate curve for the borrowed token. Separating the rate curves gives governance more fine-grained control over supply rates and reserves, the downside is that it requires twice as much work from governance to maintain. However, since Comet only supports borrowing and supply rates in the stable coin market, the management complexity is still less than v2.

Balances, Principal and Indices

A multiplicative index can be calculated as:

$$Index_{T_1} = Index_{T_0}(T_1 - T_0)$$

where $Index_T$ represents the interest index at time T, and T_N represents the wall clock time (e.g. as a Unix epoch). Multiplicative indices are usually applied as $Balance_{T_1} = Balance_{T_0} \cdot \frac{Index_{T_1}}{Index_{T_0}}$. That is, the ratio of two indices is a multiplicative factor which correctly moves a balance forward in time with interest. This is the same method used in Compound v2 *C-Tokens*.

In Comet, interest-bearing balances are represented internally as *principal* amounts. A principal, derived from a balance at time T, is a new balance, such that if you accrued interest from the beginning of time, that balance's current vaue, at time T, would equal the original given balance. That is, it is what a balance would have been at T_0 to be equal in value today. This principal value can be easily derived for a $Balance_{T_N}$ by taking $Principal = Balance_{T_N} \cdot \frac{Index_{T_0}}{Index_{T_N}}$. The alternative would be to store the tuple $(Balance_{T_N}, T_N)$, but we can save space by storing the singular *principal*. We sometimes refer to principals as day-zero balances, since they correspond to the balance as of the first index, $Index_{T_0}$.

Indices are always assumed to be unsigned integers, while balances are signed integers. This is in contrast to Compound v2 where both indices and balances were unsigned. However in v2, balances were separated into supply and borrow amounts, whereas here they are combined into a single signed number (and thus supply and borrow cannot be maintained at the same time, for an account balance of any asset).

Factors

A *factor* through this document refers to a fixed-digit decimal number. Specifically, a decimal number scaled by 1e18. These numbers should be treated as real numbers scaled down by 1e18. For example, the number 50% would be represented as $0.5e18$ and stored in binary as $0x4563918244f40000$.

* Note: not to be confused with $CollateralFactor$ values, which may themselves, be stored as *factors*.

Protocol Contract

Configuration Constants

Configuration Constants

Name	Type	Description
$\mathbb{G}$overnor	*address*	The governor of the protocol.
$\mathbb{P}$rice$\mathbb{O}$racle	*address*	Address of the price oracle.
$\mathbb{B}$ase$\mathbb{T}$oken	*address*	Address of the base token.
$\mathbb{C}$ollateral$\mathbb{A}$ssets	*address*[]	The list of collateral asset addresses.
$\mathbb{B}$orrow$\mathbb{C}$ollateral$\mathbb{F}$actor$_{Asset}$	$factor$	Collateral factor for given asset required in order to initiate a borrow.
$\mathbb{L}$iquidate$\mathbb{C}$ollateral$\mathbb{F}$actor$_{Asset}$	$factor$	Collateral factor for given asset used when performing liquidity checks. Greater than the $\mathbb{B}$orrow$\mathbb{C}$ollateral$\mathbb{F}$actor$_{Asset}$ to avoid excessive liquidation.
$\mathbb{L}$iquidation$\mathbb{P}$enalty$_{Asset}$	$factor$	Fraction of collateral value received in borrow token when liquidated.
$\mathbb{S}$tore$\mathbb{F}$ront$\mathbb{D}$iscount$\mathbb{F}$actor$_{Asset}$	$factor$	Factor to multiply by when calculating the store-front collateral price.
$\mathbb{T}$arget$\mathbb{R}$eserves	*uint*	Minimum borrow token reserves which must be held before collateral is hodled.
$\mathbb{A}$bsorb$\mathbb{T}$ip	*uint*	*TO BE DEFINED*
$\mathbb{B}$orrow$\mathbb{M}$in	*uint*	The minimum borrow amount required to enter into a borrow position.
$\mathbb{S}$upply$\mathbb{C}$ap$_{Asset}$	*uint*	Maximum supply of asset which is allowed to be supplied.
$\mathbb{B}$ase$\mathbb{T}$racking$\mathbb{S}$upply$\mathbb{S}$peed	$factor$	Speed to track per second for suppliers.

Name	Type	Description
BaseTrackingBorrowSpeed	*factor*	Speed to track per second for borrowers.
SupplyRateBase	*factor*	Base rate for supply.
SupplyRateSlope	*factor*	Slope of supply rate curve as a function of utilization.
BorrowRateBase	*factor*	Base rate for borrow.
BorrowRateSlope	*factor*	Slope of borrow rate curve as a function of utilization.

Storage

Name	Type	Description
TotalSupplyBase	*uint*72	Total amount of base token principal which the protocol owes to suppliers.
TotalBorrowBase	*uint*72	Total amount of base token principal which borrowers owe to the protocol.
LastAccrualTime	*uint*48	Timestamp of last interest accrual.*Note*: Split storage between 2 slots with 24-bits available each.
BaseSupplyIndex	*uint*64	Interest index for base token supply principal.
BaseBorrowIndex	*uint*64	Interest index for base token borrow principal.
TrackingSupplyIndex	*uint*96	Index tracking total protocol participation for supply.
TrackingBorrowIndex	*uint*96	Index tracking the total protocol partipcation for borrows.
IsPermitted$_{Owner,\ Manager}$	*bool*	Whether or not the *Manager* has permission to manage the *Owner* account.
UserPrincipal$_{Account}$	*int*72	Amount of stable coin principal which is owed to a given account (+) or by it (-).
UserBaseTrackingIndex$_{Account}$	*uint*96	The index tracking user participation for a given account.

Name	Type	Description
UserBaseTrackingAccrued$_{Account}$	$uint48$	Total participation tracking index previously earned by an account.
TotalCollateral$_{Asset}$	$uint128$	Total amount of given collateral asset which the protocol owes to borrowers.
CollateralTrackingIndex$_{Asset}$	$uint128$	The global tracking index for an asset. [TBD]
UserCollateral$_{Asset,\ Account}$	$uint128$	Amount of given collateral asset owed to a given account.
UserCollateralTrackingIndex$_{Asset,\ Account}$	$uint128$	The collateral tracking index for an asset as of the last balance interaction by an account. [TBD]

> **TODO**: We're pretty much going to need implicit assets you're in, otherwise liquidity checks are too expensive * 16-bit vector stored with UserStable? [name=Jared] [time=Mon, Nov 8, 2021 10:00 PM] [color=blue]

Constructor

Constructor()

- **Write** LastAccrualTime $= System_{Now}$
- **Write** BaseSupplyIndex $= 1.0$
- **Write** BaseBorrowIndex $= 1.0$
- **Write** TrackingSupplyIndex $= 1.0$
- **Write** TrackingBorrowIndex $= 1.0$

Account Functions

Allow(Owner, Manager, IsAllowed) [External] Allow or disallow another address to withdraw, or transfer from the Sender address.

- **Write** IsPermitted$_{Owner,\ Manager}$ $= \mathsf{IsAllowed}$

Supply(Asset, Amount) [External]

- **Call** $Supply(\mathsf{Sender}, \mathsf{Asset}, \mathsf{Amount})$

Supply(Dst, Asset, Amount) [External]

- **Call** $Supply(\mathsf{Sender}, \mathsf{Dst}, \mathsf{Asset}, \mathsf{Amount})$

Supply(From, Dst, Asset, Amount) [External]

- **When** $\mathsf{Asset} = \mathbb{B}\text{ase}\mathbb{T}\text{oken}$:
  - **Call** $SupplyBase(\mathsf{Sender}, \mathsf{From}, \mathsf{Dst}, \mathsf{Amount})$
- **Else**
  - **Call** $SupplyCollateral(\mathsf{Sender}, \mathsf{From}, \mathsf{Dst}, \mathsf{Asset}, \mathsf{Amount})$

SupplyCollateral(Operator, From, Dst, Asset, Amount) [Internal] Supplies a collateral token to the protocol, which the account can later borrow against.

- **Require** $HasPermission(\mathsf{From},\ \mathsf{Operator})$
- **External Trx** $\mathbb{E}\text{rc20}(\mathsf{Asset}).\,\text{transfer}F\text{rom}(\mathsf{From}, Contract_{This}, \mathsf{Amount})$
  - Let *txAmount* be the actual amount transferred less any fees.
- **Write** $\mathfrak{T}\text{otal}\mathfrak{C}\text{ollateral}_{\mathsf{Asset}} \mathrel{+}= \mathit{txAmount}$
- **Write** $\mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{\mathsf{Asset},\ \mathsf{Dst}} \mathrel{+}= \mathit{txAmount}$
- **Require** $\mathfrak{T}\text{otal}\mathfrak{C}\text{ollateral}_{\mathsf{Asset}} \leq \mathbb{S}\text{upply}\mathbb{C}\text{ap}_{\mathsf{Asset}}$

SupplyBase(Operator, From, Dst, Amount) [Internal] Transfers in borrow token pegged to the user's account. This will repay any outstanding borrows before adding to a user's supply. If the user has a positive supply balance, their accont will receive yield along the supply curve.

- **Require** $HasPermission(\mathsf{From},\ \mathsf{Operator})$
- **External Trx** $\mathbb{E}\text{rc20}(\mathsf{Asset}).\,\text{transfer}F\text{rom}(\mathsf{From}, Contract_{This}, \mathsf{Amount})$
  - Let *txAmount* be the actual amount transferred less any fees.
- **Call** $\mathit{Accrue}()$
- **Read** $\mathit{dstPrincipal} = \mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Dst}}$
- Let $\mathit{dstBalance} = PresentValue(\mathit{dstPrincipal})$
- **Read** $\mathit{totalSupplyBaseBalance} = PresentValue_{Supply}(\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase})$
- **Read** $\mathit{totalBorrowBaseBalance} = PresentValue_{Borrow}(\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase})$
- Let $\mathit{repaySupply} = \mathit{RepayAndSupplyAmount}(\mathit{dstBalance},\ \mathit{txAmount})$
- Let $\mathit{dstBalance}' = \mathit{dstBalance} + \mathit{txAmount}$
- Let $\mathit{totalSupplyBaseBalance}' = \mathit{totalSupplyBaseBalance} + \mathit{repaySupply}_{\mathit{supply}}$
- Let $\mathit{totalBorrowBaseBalance}' = \mathit{totalBorrowBaseBalance} - \mathit{repaySupply}_{\mathit{repay}}$
- **Call** $\mathit{UpdateBaseBalance}(\mathsf{Dst},\ \mathit{dstPrincipal},\ PrincipalValue(\mathit{dstBalance}'))$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase} = PrincipalValue_{Supply}(\mathit{totalSupplyBaseBalance}')$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase} = PrincipalValue_{Borrow}(\mathit{totalBorrowBaseBalance}')$

Transfer(Dst, Asset, Amount) [External]

- **Call** $Transfer(\mathsf{Sender}, \mathsf{Dst}, \mathsf{Asset}, \mathsf{Amount})$

Transfer(Src, Dst, Asset, Amount) [External]

- **When** $\mathsf{Asset} = \mathbb{B}\text{ase}\mathbb{T}\text{oken}$:
  - **Call** $TransferBase(\mathsf{Sender}, \mathsf{Src}, \mathsf{Dst}, \mathsf{Amount})$

- **Else**
    - **Call** $TransferCollateral(\mathsf{Sender}, \mathsf{Src}, \mathsf{Dst}, \mathsf{Asset}, \mathsf{Amount})$

TransferCollateral(Operator, Src, Dst, Asset, Amount) [Internal] Transfers collateral between users. Reverts if the Src user would have negative liquidity after the transfer.

- **Require** $HasPermission(\mathsf{Src},\ \mathsf{Operator})$
- **Write** $\mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{\mathsf{Asset},\ \mathsf{Src}} -= \mathsf{Amount}$
- **Write** $\mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{\mathsf{Asset},\ \mathsf{Dst}} += \mathsf{Amount}$
- **Require** $IsBorrowCollateralized(\mathsf{Src})$
    - *Note*: We don't need to accrue interest since $BorrowCF < LiquidationCF$ covers small changes

TransferBase(Operator, Src, Dst, Amount) [Internal] Transfers base token between accounts. Reverts if Src account would have negative liquidity after the transfer.

- **Require** $HasPermission(\mathsf{Src},\ \mathsf{Operator})$
- **Call** $Accrue()$
- **Read** $srcPrincipal = \mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Src}}$
- **Read** $dstPrincipal = \mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Dst}}$
- Let $srcBalance = PresentValue(srcPrincipal)$
- Let $dstBalance = PresentValue(dstPrincipal)$
- **Read** $totalSupplyBaseBalance = PresentValue_{Supply}(\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase})$
- **Read** $totalBorrowBaseBalance = PresentValue_{Borrow}(\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase})$
- Let $withdrawBorrow = WithdrawAndBorrowAmount(srcBalance,\ \mathsf{Amount})$
- Let $repaySupply = RepayAndSupplyAmount(dstBalance,\ \mathsf{Amount})$
- Let $srcBalance' = srcBalance - \mathsf{Amount}$
- Let $dstBalance' = dstBalance + \mathsf{Amount}$
- Let $totalSupplyBaseBalance' = totalSupplyBaseBalance + repaySupply_{supply} - withdrawBorrow_{withdraw}$
- Let $totalBorrowBaseBalance' = totalBorrowBaseBalance + withdrawBorrow_{borrow} - repaySupply_{repay}$
- **Call** $UpdateBaseBalance(\mathsf{Src},\ srcPrincipal,\ PrincipalValue(srcBalance'))$
- **Call** $UpdateBaseBalance(\mathsf{Dst},\ dstPrincipal,\ PrincipalValue(dstBalance'))$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase} = PrincipalValue_{Supply}(totalSupplyBaseBalance')$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase} = PrincipalValue_{Borrow}(totalBorrowBaseBalance')$
- If $srcBalance' < 0$
    - **Require** $|srcBalance'| \geq \mathbb{B}\text{orrow}\mathbb{M}\text{in}$

Withdraw(Asset, Amount) [External]

- **Call** $Withdraw(\mathsf{Sender}, \mathsf{Asset}, \mathsf{Amount})$

Withdraw(To, Asset, Amount) [External]

- **Call** $Withdraw(\mathsf{Sender}, \mathsf{To}, \mathsf{Asset}, \mathsf{Amount})$

Withdraw(Src, To, Asset, Amount) [External]

- **When** $\mathsf{Asset} = \mathbb{B}\text{ase}\mathbb{T}\text{oken}$:
  - **Call** $WithdrawBase(\mathsf{Sender}, \mathsf{Src}, \mathsf{To}, \mathsf{Amount})$
- **Else**
  - **Call** $WithdrawCollateral(\mathsf{Sender}, \mathsf{Src}, \mathsf{To}, \mathsf{Asset}, \mathsf{Amount})$

WithdrawCollateral(Operator, Src, To, Asset, Amount) [Internal] Transfers out collateral from the Sender account to the To account. Reverts if the caller would have negative liquidity after withdrawal.

- **Require** $HasPermission(\mathsf{Src},\ \mathsf{Operator})$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{C}\text{ollateral}_{\mathsf{Asset}} -= \mathsf{Amount}$
- **Write** $\mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{\mathsf{Asset},\ \mathsf{Src}} -= \mathsf{Amount}$
- **Require** $IsBorrowCollateralized(\mathsf{Src}) \lor \mathsf{Sender} = Contract_{This}$
  - *Note*: Primary conditional allows selling reclaimed collateral while underwater.
- **External Trx** $\mathbb{E}\text{rc20}(\mathsf{Asset}).\,\text{transfer}(\mathsf{To}, \mathsf{Amount})$

WithdrawBase(Operator, Src, To, Amount) [Internal] Transfers out base token from the Sender account to the To account. Reverts if the caller would have negative liquidity after withdrawal.

- **Require** $HasPermission(\mathsf{Src},\ \mathsf{Operator})$
- **Call** $\mathit{Accrue}()$
- **Read** $\mathit{srcPrincipal} = \mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Src}}$
- Let $\mathit{srcBalance} = PresentValue(\mathit{srcPrincipal})$
- **Read** $\mathit{totalSupplyBaseBalance} = PresentValue_{Supply}(\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase})$
- **Read** $\mathit{totalBorrowBaseBalance} = PresentValue_{Borrow}(\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase})$
- Let $\mathit{withdrawBorrow} = WithdrawAndBorrowAmount(\mathit{srcBalance},\ \mathsf{Amount})$
- Let $\mathit{srcBalance}' = \mathit{srcBalance} - \mathsf{Amount}$
- Let $\mathit{totalSupplyBaseBalance}' = \mathit{totalSupplyBaseBalance} - \mathit{withdrawBorrow}_{withdraw}$
- Let $\mathit{totalBorrowBaseBalance}' = \mathit{totalBorrowBaseBalance} + \mathit{withdrawBorrow}_{borrow}$
- **Call** $\mathit{UpdateBaseBalance}(\mathsf{Src},\ \mathit{srcPrincipal},\ \mathit{PrincipalValue}(\mathit{srcBalance}'))$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase} = PrincipalValue_{Supply}(\mathit{totalSupplyBaseBalance}')$
- **Write** $\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase} = PrincipalValue_{Borrow}(\mathit{totalBorrowBaseBalance}')$
- If $\mathit{srcBalance}' < 0$
  - **Require** $|\mathit{srcBalance}'| \geq \mathbb{B}\text{orrow}\mathbb{M}\text{in}$
- **Require** $IsBorrowCollateralized(\mathsf{Src})$
- **External Trx** $\mathbb{E}\text{rc20}(\mathbb{B}\text{ase}\mathbb{T}\text{oken}).\,\text{transfer}(\mathsf{To})\mathsf{Amount}$

Interest and Tracking Functions

Accrue() [Internal] Accrue interest in base token supply and borrows. This function also tracks participation in the protocol.

```
• Read $timeElapsed = System_{Now} - \mathcal{L}ast\mathcal{A}ccrual\mathcal{T}ime$
• When $timeElapsed > 0$:
    – Write $\mathcal{B}ase\mathcal{S}upply\mathcal{I}ndex \mathrel{+}= \mathcal{B}ase\mathcal{S}upply\mathcal{I}ndex \cdot GetSupplyRate() \cdot timeElapsed$
    – Write $\mathcal{B}ase\mathcal{B}orrow\mathcal{I}ndex \mathrel{+}= \mathcal{B}ase\mathcal{B}orrow\mathcal{I}ndex \cdot GetBorrowRate() \cdot timeElapsed$
    – Write $\mathcal{T}racking\mathcal{S}upply\mathcal{I}ndex \mathrel{+}= \frac{\mathbb{B}ase\mathbb{T}racking\mathbb{S}upply\mathbb{S}peed}{\mathcal{T}otal\mathcal{S}upply\mathcal{B}ase} \cdot timeElapsed$
    – Write $\mathcal{T}racking\mathcal{B}orrow\mathcal{I}ndex \mathrel{+}= \frac{\mathbb{B}ase\mathbb{T}racking\mathbb{B}orrow\mathbb{S}peed}{\mathcal{T}otal\mathcal{B}orrow\mathcal{B}ase} \cdot timeElapsed$
    – Write $\mathcal{L}ast\mathcal{A}ccrual\mathcal{T}ime = System_{Now}$
```

UpdateBaseBalance(Account, InitialUserBalance, FinalUserBalance) [Internal] Write updated balance to store and tracking participation.

```
• When $\mathsf{InitialUserBalance} \geq 0$:
    – Read $indexDelta = \mathcal{T}racking\mathcal{S}upply\mathcal{I}ndex - \mathcal{U}ser\mathcal{B}ase\mathcal{T}racking\mathcal{I}ndex_{\mathsf{Account}}$
• Otherwise
    – Read $indexDelta = \mathcal{T}racking\mathcal{B}orrow\mathcal{I}ndex - \mathcal{U}ser\mathcal{B}ase\mathcal{T}racking\mathcal{I}ndex_{\mathsf{Account}}$
• When $\mathsf{FinalUserBalance} \geq 0$:
    – Write $\mathcal{U}ser\mathcal{B}ase\mathcal{T}racking\mathcal{I}ndex_{\mathsf{Account}} = \mathcal{T}racking\mathcal{S}upply\mathcal{I}ndex$
• Otherwise
    – Write $\mathcal{U}ser\mathcal{B}ase\mathcal{T}racking\mathcal{I}ndex_{\mathsf{Account}} = \mathcal{T}racking\mathcal{B}orrow\mathcal{I}ndex$
• Write $\mathcal{U}ser\mathcal{B}ase\mathcal{T}racking\mathcal{A}ccrued_{Account} \mathrel{+}= \mathsf{InitialUserBalance} \cdot indexDelta$
```

GetSupplyRate(): factor [External] Return the current supply rate.

```
• Let $utilization = GetUtilization()$
• Return $\mathbb{S}upply\mathbb{R}ate\mathbb{B}ase + utilization \times \mathbb{S}upply\mathbb{R}ate\mathbb{S}lope$
```

GetBorrowRate(): factor [External] Return the current borrow rate.

```
• Let $utilization = GetUtilization()$
• Return $\mathbb{B}orrow\mathbb{R}ate\mathbb{B}ase + utilization \times \mathbb{B}orrow\mathbb{R}ate\mathbb{S}lope$
```

GetUtilization(): factor [External] Returns the current protocol utilization.

```
• Read $totalSupply = PresentValue(\mathcal{T}otal\mathcal{S}upply\mathcal{B}ase)$
• Read $totalBorrows = PresentValue(\mathcal{T}otal\mathcal{B}orrow\mathcal{B}ase)$
• When $totalSupply = 0$:
    – Return $0$
• Otherwise
    – Return $totalBorrows \div totalSupply$
```

Do we need to factor in reserves here? That could be an expensive extra operation.

> No I don't think so, this looks good - we used to use cash and reserves to get supply [name=Jared] [color=yellow]

Liquidation Functions

Absorb(Account) [External] Transfer user's debt to protocol accounts, decreasing cash reserves and adding collateral to the protocol's own balance. The caller is given an absorption incentive.

- **Require** $IsLiquidatable(\textsf{Account})$
- **Read** $acctPrincipal = \mathfrak{U}ser\mathfrak{P}rincipal_{\textsf{Account}}$
- Let $basePrice = GetPrice(\mathbb{B}ase\mathbb{T}oken)$:new: :male-cook:
- Let $acctBalance = PresentValue(acctPrincipal)$
- Initialize $acctBalance' = acctBalance$
- For $asset \in \mathbb{C}ollateral\mathbb{A}ssets$ `# TODO: Assets you're in?`
  - **Read** $seizeAmount = \mathfrak{U}ser\mathfrak{C}ollateral_{asset,\ \textsf{Account}}$
  - If $seizeAmount > 0$:
    - **Write** $\mathfrak{U}ser\mathfrak{C}ollateral_{asset,\ \textsf{Account}} -= seizeAmount$
      - TODO: is this always eq 0?
    - **Write** $\mathfrak{U}ser\mathfrak{C}ollateral_{asset,\ Contract_{This}} += seizeAmount$
      - TODO: liq dao?
    - $acctBalance' += seizeAmount \times GetPrice(asset) \cdot \mathbb{L}iquidation\mathbb{P}enalty_{asset}$
- $acctBalance' = \frac{max(acctBalance',0)}{basePrice}$:new: :male-cook:
  - TODO: Should we track any deficit here?
- **Write** $\mathfrak{U}ser\mathfrak{P}rincipal_{\textsf{Account}} = PrincipalValue(acctBalance')$
  - TODO: Why don't we use UpdateBaseBalance here?
- **Write** $\mathfrak{T}otal\mathfrak{S}upply\mathfrak{B}ase += PrincipalValue_{Supply}(acctBalance')$
- **Write** $\mathfrak{T}otal\mathfrak{B}orrow\mathfrak{B}ase -= PrincipalValue_{Borrow}(|acctBalance|)$
- Let $absorptionIncentive =$ total gas of transaction$(BASEFEE + \mathbb{A}bsorb\mathbb{T}ip)$
- **External Trx** $\mathbb{E}rc20(\mathbb{B}ase\mathbb{T}oken).\mathrm{transfer}(\textsf{Sender}, absorptionIncentive)$

XXX: Fix Absorption incentive We need gas of absorb not the transaction, we should be able to make a good/safe over-estimate, assuming we cap what external calls can cost

Absorb(Accounts) [External] Absorb multiple accounts at once.

- For $account \in \textsf{Accounts}$:
  - Call $Absorb(account)$

AskPrice(Asset, Amount) [External] Calculate the store-front price for a given amount of collateral for sale. Does not check if the quantity is actually available for sale.

- Return $GetPrice(\textsf{Asset}) \cdot \mathbb{S}tore\mathbb{F}ront\mathbb{D}iscount\mathbb{F}actor_{\textsf{Asset}}$

BuyCollateral(Asset, Amount, BaseAmount) Buy collateral from the protocol using base tokens, increasing reserves. A minimum collateral amount should be specified to indicate the maximum slippage acceptable for the buyer.

Note: we choose to implement a simple auction strategy which seemed to do well in simulations, this is a likely point for experimentation within the protocol.

```
• When $GetReserves() < \mathbb{T}\text{arget}\mathbb{R}\text{eserves}$:
    – Read $collateralAmount = \frac{\mathsf{BaseAmount}}{AskPrice(Asset, Amount)}$
    – Require $collateralAmount \geq MinCollateralAmount$
    – Call $SupplyReserves(\mathsf{Sender}, \mathsf{Sender}, BaseAmount)$
    – Call $WithdrawCollateral(Contract_{This}, Contract_{This}, Recipient, Asset, collateralAmount)$
```

Reserves Functions

__WithdrawReserves(To, Amount) [Internal] Withdraw reserves from the protocol to another account.

```
• Require $\mathsf{Sender} = \mathbb{G}\text{overnor}$
• External Trx $\mathbb{E}\text{rc20}(\mathbb{B}\text{ase}\mathbb{T}\text{oken}).\text{transfer}(\mathsf{To})\mathsf{Amount}$
```

GetReserves(): int [External]

```
• External Call $thisBalance = \mathbb{E}\text{rc20}(\mathbb{B}\text{ase}\mathbb{T}\text{oken}).\text{balance}Of(Contract_{This})$
• Return $thisBalance - PresentValue_{Supply}(\mathfrak{T}\text{otal}\mathfrak{S}\text{upply}\mathfrak{B}\text{ase}) + PresentValue_{Borrow}(\mathfrak{T}\text{otal}\mathfrak{B}\text{orrow}\mathfrak{B}\text{ase})$
```

Helper Functions

IsBorrowCollateralized(Account): bool [External] Returns true if the account has non-negative liquidity using the borrow collateral factors.

```
• Read $liquidity = -GetPrice(\mathbb{B}\text{ase}\mathbb{T}\text{oken}) \times PresentValue(\mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Account}})$
  :new: :male-cook:
• For $asset \in \mathbb{C}\text{ollateral}\mathbb{A}\text{ssets}$
    – If $liquidity \geq 0$
        ∗ Return $true$
    – $liquidity \mathrel{+}= \mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{asset,\ \mathsf{Account}} \cdot GetPrice(\mathsf{Asset}) \cdot \mathbb{B}\text{orrow}\mathbb{C}\text{ollateral}\mathbb{F}\text{actor}_{\mathsf{Asset}}$
• Return $liquidity \geq 0$
```

IsLiquidatable(Account): bool [External] Returns true if the account has negative liquidity using the liquidation collateral factors.

```
• Read $liquidity = -GetPrice(\mathbb{B}\text{ase}\mathbb{T}\text{oken}) \times PresentValue(\mathfrak{U}\text{ser}\mathfrak{P}\text{rincipal}_{\mathsf{Account}})$
  :new: :male-cook:
• For $asset \in \mathbb{C}\text{ollateral}\mathbb{A}\text{ssets}$
    – If $liquidity \geq 0$
        ∗ Return $true$
```

  - $liquidity \mathrel{+}= \mathfrak{U}\text{ser}\mathfrak{C}\text{ollateral}_{asset,\ \mathsf{Account}} \cdot GetPrice(\mathsf{Asset}) \cdot \mathbb{L}\text{iquidate}\mathbb{C}\text{ollateral}\mathbb{F}\text{actor}_{\mathsf{Asset}}$
- **Return** $liquidity \geq 0$

GetPrice(Asset): factor [External] Get the price of an asset.

HasPermission(address Owner, address Manager): bool [Internal]

- **Return** $\mathsf{Owner} = \mathsf{Manager} \vee \mathfrak{I}\text{s}\mathfrak{P}\text{ermitted}_{\mathsf{Owner},\ \mathsf{Manager}}$

PrincipalValue(int PresentValue): int [Internal] Return the positive principal supply balance if positive or the negative borrow balance if negative.

- If $\mathsf{PresentValue} \geq 0$:
  - **Return** $PrincipalValue_{Supply}(\mathsf{PresentValue})$
- Else:
  - **Return** $PrincipalValue_{Borrow}(\mathsf{PresentValue})$

PrincipalValueSupply(uint PresentValue): uint [Internal] Return the amount projected backward by the supply index. * **Read and Return** $\frac{\mathsf{PresentValue}}{\mathfrak{B}\text{ase}\mathfrak{S}\text{upply}\mathfrak{I}\text{ndex}}$

PrincipalValueBorrow(uint PresentValue): uint [Internal] Return the amount projected backward by the borrow index.

- **Read and Return** $\frac{\mathsf{PresentValue}}{\mathfrak{B}\text{ase}\mathfrak{B}\text{orrow}\mathfrak{I}\text{ndex}}$

PresentValue(int PrincipalValue): int [Internal] Return the positive present supply balance if positive or the negative borrow balance if negative.

- If $\mathsf{PrincipalValue} \geq 0$:
  - **Return** $PresentValue_{Supply}(\mathsf{PrincipalValue})$
- Else
  - **Return** $PresentValue_{Borrow}(\mathsf{PrincipalValue})$

PresentValueSupply(uint PrincipalValue): uint [Internal] Return the principal amount projected forward by the supply index.

- **Read and Return** $\mathsf{PrincipalValue} \cdot \mathfrak{B}\text{ase}\mathfrak{S}\text{upply}\mathfrak{I}\text{ndex}$

PresentValueBorrow(uint PrincipalValue): uint [Internal] Return the principal amount projected forward by the borrow index.

- **Read and Return** $\mathsf{PrincipalValue} \cdot \mathfrak{B}\text{ase}\mathfrak{B}\text{orrow}\mathfrak{I}\text{ndex}$

RepayAndSupplyAmount(int Balance, uint Amount): (uint, uint) [Internal]

- Let $repayAmount = max(min(-\mathsf{Balance}, \mathsf{Amount}), 0)$
- Let $supplyAmount = \mathsf{Amount} - repayAmount$
- **Return** $\{repay = repayAmount, supply = supplyAmount\}$

WithdrawAndBorrowAmount(int Balance, uint Amount): (uint, uint) [Internal]

- Let $withdrawAmount = max(min(\mathsf{Balance}, \mathsf{Amount}), 0)$
- Let $borrowAmount = \mathsf{Amount} - withdrawAmount$
- **Return** $\{withdraw = withdrawAmount, borrow = borrowAmount\}$

Liquidation

When an account goes underwater, its position can be absorbed into the protocol account, buying all the collateral belonging to the position in exchange for paying down their debt. The protocol then attempts to sell off the collateral in order to recover reserves which have been paid out to accounts in this way.

Tracking

The protocol tracks participation in markets and accrues that to each account. This allows external contracts to confidently pull information about how long an account has participated in the Compound Protocol and how much value that account has provided to the protocol, in general.

Optional or External Helpers

This section is incomplete.

TransferStableMaxWithoutBorrowing(Operator, Src, Dst)

TransferStableMaxWithBorrowing(Operator, Src, Dst)

WithdrawStableMaxWithoutBorrowing(Operator, Src, To)

WithdrawStableMaxWithBorrowing(Operator, Src, To)

XXXAndCall(Operator, . . .)

GaslessSigning