Compound Protocol

[RELEASE CANDIDATE] Version 1.0

Parameters

Constant	Description
collateralRatio	The amount of supply value required to be held for each unit of borrow value to initiate an action (borrow, withdraw), must be strictly greater than 1, e.g. 2.0.
liquidationDiscount	A multiplier representing the percent value that a user calling liquidate receives, e.g. 0.05 for 5%
borrowFee	The percent fee that a user originating a borrow has added to their balance, e.g. 0.0005.

Value	Description
supplyCurrent	The user's supply of a given asset, including accrued interest as of the current block
borrowCurrent	The user's borrow of a given asset, including accrued interest as of the current block
accountLiquidity	The value of a user's account, denominated in Ether, above the user's collateral requirement
withdrawCapacity	The quantity of tokens, denominated in the specified asset, that the user is able to withdraw from Compound, or accountLiquidity divided by the oracle price of the asset.
borrowCapacity	The quantity of tokens, denominated in the specified asset, that the user is able to borrow from Compound
shortfall	The amount of value, denominated in Ether, that a target user can have seized
amountClose	The amount of asset that a target user can have closed
amountSeize	The amount of collateral, denominated in its own asset, to seize from a target user
totalSupply	The sum of supply of an asset in its respective money market, excluding accrued interest

totalBorrow	The sum of borrow of an asset in its respective money market, excluding accrued interest

Exceptional States

We assume that in *any* error condition, either a) the protocol exits gracefully with an event describing the error if no side-effects have yet occured, or b) the transaction fails completely. Any exceptions to this rule are noted.

Interest Rate Model

For each asset, there are two *interest rate indices*: one for supply, one for borrow. Each index tracks the growth of a user's balance given a start time, and an end time, and implicitly tracks the history of interest rates over time. We assert that the interest rate model is a pure function over the supply, borrow and cash of an asset in the market.

Any time the utilization rate of an asset changes, we update **both** interest rate indices.

Price Oracle

The Compound Protocol uses interest rates which are stored in a Price Oracle, which is a satellite smart contract. The Withdraw, Borrow, and Liquidate functions all reference the prices in the Oracle.

See: Oracle Specification

Supply Asset(Address asset, Amount supplyAmount)

Users supply assets from their own address into the protocol.

- Fail if market not supported
- Fail gracefully if asset is not approved or has insufficient balance
- We calculate the user's supplyCurrent for the asset
    - To calculate the currentSupplyIndex value (as of the transaction's block), we multiply the most recent supply index by $1 + blocks \times rate$, the interest accrued since the last index snapshot
    - The user's last balance checkpoint is multiplied by the currentSupplyIndex value and divided by the user's checkpoint index value
- We ERC-20 *transfer* the asset into the protocol (*note: pre-conditions already checked above)*
- We update the protocol's totalSupply by subtracting the user's prior checkpointed balance, adding supplyCurrent, and supplyAmount
    - The utilization rate has changed (due to cash increasing)! We calculate a new supply index, new supply rate, new borrow index, and new borrow rate for the asset, and save all four.
- We checkpoint the user's new balance, supplyCurrent + supplyAmount at the updated supply index

Withdraw Asset(Address asset, Amount withdrawAmount)

Users withdraw assets from the protocol into their own address.

- We calculate the user's supplyCurrent for the asset
    - To calculate the currentSupplyIndex value (as of the transaction's block), we multiply the most recent supply index by $(1 + blocks \times rate)$, or the interest accrued since the last index snapshot
    - The user's last balance checkpoint is multiplied by the currentSupplyIndex value, divided by the user's checkpoint index value
- We calculate the user's accountLiquidity
    - $accountLiquidity = \sum_{a \in assets} oracle_a \cdot (supplied_a - collateralRatio \cdot borrowed_a)$
    - We get each of these balances, supplies and borrows, by taking their last checkpoint, and reading the current supply or borrow index dividing it by their checkpoint index and multiplying it by their checkpoint balance.
    - To calculate the current supply or borrow index value (as of the transaction's block), we multiply the most recent supply or borrow index by $1 + blocks \times rate$, the interest accrued since the last index snapshot
- We calculate the user's withdrawCapacity, denominated in the asset.
  Given:
    - $withdrawCapacity = accountLiquidity / oracle_a$

  We re-arrange terms:
    - $withdrawCapacity * oracle_a = accountLiquidity$
    - Since $withdrawAmount <= withdrawCapacity$ it follows:
    - $withdrawAmount * oracle_a <= accountLiquidity$
    - In other words:
    - $ethValueOfWithdrawAmount <= accountLiquidity$
- If the user specifies *-1* amount to withdraw ("max"), withdrawAmount => the lesser of withdrawCapacity and supplyCurrent
- We check that the amount is less than withdrawCapacity, and less than or equal to supplyCurrent
- Fail gracefully if protocol has insufficient cash
- We update the protocol's totalSupply, by subtracting the user's prior checkpointed balance, adding supplyCurrent, and subtracting withdrawAmount
    - The utilization rate has changed! We calculate a new supply index and borrow index for the asset, and save it.
    - Note: we previously calculated (but discard) updated balances and index values above.

- We checkpoint the user’s new supply balance, supplyCurrent - withdrawAmount at the updated index
- We ERC-20 *transfer* the asset out of the protocol to the user

Borrow Asset(Address asset, Amount borrowAmount)

Users borrow assets from the protocol into their own address.

- Fail if market not supported
- We calculate the user's borrowCurrent for the asset
  - To calculate the currentBorrowIndex value (as of the transaction's block), we multiply the most recent borrow index by $(1 + blocks \times rate)$, or the interest accrued since the last index snapshot
  - The user's last balance checkpoint is multiplied by the currentBorrowIndex value, divided by the user's checkpoint index value
- We calculate the user's accountLiquidity
  - $accountLiquidity = \sum_{a \in assets} oracle_a \cdot (supplied_a - collateralRatio \cdot borrowed_a)$
  - We get each of these balances, supplies and borrows, by taking their last checkpoint, and reading the current supply or borrow index dividing it by their checkpoint index and multiplying it by their checkpoint balance.
  - To calculate the current supply or borrow index value (as of the transaction's block), we multiply the most recent supply or borrow index by $1 + blocks \times rate$, the interest accrued since the last index snapshot
- We calculate the user's borrowCapacity, denominated in the asset
  - $borrowCapacity = accountLiquidity / [(collateralRatio * oracle_a) * (1 + borrowFee)]$
  -
- We check that borrowAmount is less than borrowCapacity.
  - $[(collateralRatio * oracle_a * borrowAmount) * (1 + borrowFee)] < accountLiquidity$
- Fail gracefully if protocol has insufficient cash
- We ERC-20 *transfer* the asset out.
- We update the protocol's totalBorrow, by subtracting the user's prior checkpointed balance, adding borrowCurrent, and adding borrowAmount * (1+ borrowFee)
  - The utilization rate has changed! We calculate a new supply index and borrow index for the asset, and save it.
- We checkpoint the user's new borrow balance, borrowCurrent + borrowAmount * (1+ borrowFee) at the updated index
- We ERC-20 *transfer* the borrowAmount of asset out of the protocol to the user
  - Note: this should only fail if the protocol has insufficient cash

Repay Borrow(Address asset, Amount repayAmount)

Users repay borrowed assets from their own address to the protocol.

- We calculate the user's borrowCurrent for the asset
  - To calculate the currentBorrowIndex value (as of the transaction's block), we multiply the most recent borrow index by $1 + blocks \times rate$, or the interest accrued since the last index snapshot
  - The user's last balance checkpoint is multiplied by the currentBorrowIndex value, divided by the user's checkpoint index value
- If the user specifies *-1* amount to repay ("max"), repayAmount => the lesser of the senders ERC-20 balance and borrowCurrent
- We check that repayAmount is less than borrowCurrent and less than or equal to their ERC-20 balance
- Fail gracefully if asset is not approved or has insufficient balance
- We ERC-20 *transfer* the asset into the protocol
- We update the protocol's totalBorrow, by subtracting the user's prior checkpointed balance, adding borrowCurrent, and subtracting repayAmount
- The transfer in increases totalCash.
  - The utilization rate has changed! We calculate a new supply index and borrow index for the asset using the new totalBorrow and totalCash and save it.
- We checkpoint the user's new borrow balance, borrowCurrent - repayAmount at the updated index

Liquidate(Address targetUser, Address assetBorrow, Address assetCollateral, Amount requestedAmountClose)

The calling user liquidates the target user's borrow, from their own address to the protocol. The collateral seized remains as the calling user's supply inside the protocol.

- If the borrowed asset's market is Supported,
  - We calculate the target user's shortfall, denominated in Ether, that the user is below the collateral ratio:
  - $shortfall_{account} = \sum_{a \in assets} oracle_a \times (collateralRatio \cdot borrowed_a - supplied_a)$
  - *Alternatively*: $shortfall = -1 \times withdrawCapacity_{eth}$
  - We get each of these balances, supplies and borrows, by taking their last checkpoint, and reading the current supply or borrow index dividing it by their checkpoint index and multiplying it by their checkpoint balance.
  - To calculate the current supply or borrow index value (as of the transaction's block), we multiply the most recent supply or borrow index by $1 + blocks \times rate$, the interest accrued since the last index snapshot
  - If there is no shortfall, abort
- We calculate the target user's borrowCurrent for assetBorrow
  - The currentBorrowIndex value (including accrued interest as of the transaction's block) is calculated by multiplying the most recent supply index by $(1 + blocks \times rate)$ for each asset that the target user has borrowed
  - The user's last balance checkpoint is multiplied by currentBorrowIndex, divided by the user's checkpoint index value
- We calculate the target user's AND the calling user's supplyCurrent for assetCollateral
  - The currentSupplyIndex value (including accrued interest as of the transaction's block) is calculated by multiplying the most recent supply index by $(1 + blocks \times rate)$ (This is calculated once for both users.)
  - For the target user and the calling user: The user's last balance checkpoint is multiplied by currentSupplyIndex, divided by the user's checkpoint index value
- We calculate maxAmountClosable, the amount of borrow that can be closed from the target user:
  - This is equal to the lesser of:
    - borrowCurrent

            - *discountedBorrowDenominatedCollateral:* $\frac{supplyCurrent}{(1+liquiditionDiscount)} \times \frac{oracle_{assetCollateral}}{oracle_{assetBorrow}}$
            - *If market is Supported:*
                - *discountedRepayToEvenAmount:*
                    - $\frac{shortfall}{oracle_{assetBorrow} \times (collateralRatio - liquidiationDiscount - 1)}$
                - Else: ignore
- If requestedAmountClose = -1:
    - Then amountClose = maxAmountClosable
    - Else amountClose = requestedAmountClose
- Verify amountClose <= maxAmountClosable
- We calculate amountSeize;

$$amountClose \times \frac{oracle_{assetBorrow}}{oracle_{assetCollateral}} \times (1 + liquidationDiscount)$$

- Fail gracefully if asset is not approved or has insufficient balance
- We ERC-20 *transfer* amountClose of assetBorrow into Compound
- We repay the target user's borrow using the calling user's funds:
    - We update the protocol's totalBorrow for assetBorrow, by subtracting the target user's prior checkpointed balance, adding borrowCurrent, and subtracting amountClose
    - The transfer in increases totalCash.
        - The utilization rate for assetBorrow has changed! We calculate a new supply index and borrow index for the asset using the new totalBorrow and totalCash and save it.
    - We checkpoint the target user's new borrow balance, borrowCurrent - amountClose at the updated index
- We transfer the seized collateral
    - We update the protocol's totalSupply for assetCollateral, by adding the target user's supplyCurrent and subtracting their checkpointedBalance (which has the desired effect of adding accrued interest from the target user)
        - The utilization rate for assetCollateral has not changed, so we don't need to calculate new rates, but we do calculate a new supply index and borrow index for assetCollateral, and save them.
    - We checkpoint the target user's assetCollateral supply balance, supplyCurrent - amountSeize at the updated index
    - We checkpoint the calling user's assetCollateral supply balance, supplyCurrent + amountSeize at the updated index

Market States

A given asset may be in one of three states, which affect which functions are available and how the asset is utilized above.

- Initial - An asset is not part of the set “collateralAssets” nor part of “supportedAssets.” It is not used for calculating an account value and all operations, aside of “supportMarket” for that asset should fail.
- Supported - An asset is part of the sets “collateralAssets” and “supportedAssets.” It is used to calculate an account value and all operations on that asset should functional normally. The asset must have a price and interest rate model associated with it.
- Suspended - An asset is part of the set “collateralAssets” but not part of the set “supportedAssets.” Assets in this state do count for collateral, but users may only “withdraw,” “payBorrow” and “liquidate” the asset. The liquidate function no longer checks collateralization. The administrative functions below should all function as expected.

Administrative Functions

_setAdmin(Address newAdmin)

- Check caller = admin
- Store admin = newAdmin

_setMarketInterestRateModel(Address asset, Address interestRateModel)

- Check caller = admin
- Set the interest rate model to `modelAddress`

_withdrawEquity(Address asset, Amount amount)

- Check caller = admin
- equity = cash (from ERC-20 of self) + borrows - supply.
- Check that amount is less than or equal to equity
- ERC-20 Transfer cash from self to admin

_supportMarket(Address asset, Address modelAddress, Scaled price)

- Check caller = admin
- Set the asset price to `price`
- Set the interest rate model to `modelAddress`
- Default supply and borrow index to 1e18
- Append asset to collateralAssets if not set
- Set market isSupported to true

_suspendMarket(Address asset)

- Check caller = admin
- Set market isSupposed to false

_setOracle(Address Oracle)

- Check caller = admin
- oracle = Oracle

_setRiskParameters(Scaled collateralRatio, Scaled liquidationDiscount)

- Check caller = admin
- Check de-scaled collateralRatio > 1.1
- Check de-scaled 0 <= liquidationDiscount <= .1
- Check de-scaled collateralRatio > 1 + liquidationDiscount
- Set collateralRatio = new ratio
- Set liquidationDiscount = new discount

_setOriginationFee(num, denom)

- Check caller = admin
- Set origination fee = new fee

Appendix A: Discounted Repay to Even Amount

- **Prove**: "If we successfully close $discountedRepayToEvenAmount$ of an account via liquidation, then the shortfall of that account afterwards will be exactly zero."
- $Let\ C = collateralRatio, L = liquidationDiscount, P_B = oracle_{assetBorrow}, P_C = oracle_{assetCollateral}$
- $Let\ shortfall_{pre}$ be the shortfall of an account prior to liquidation
- $Let\ shortfall_{post}$ be the shortfall of an account prior to liquidation
- Let **amountClose** exactly equal: $discountedRepayToEvenAmount = \frac{shortfall_{pre}}{P_B \times (C-L-1)}$
- Thus, **amountSeize** equals:
- $\frac{shortfall_{pre}}{P_B \times (C-L-1)} \times \frac{P_B}{P_C} \times (1+L)$
- $\frac{shortfall_{pre} \times (1+L)}{P_C \times (C-L-1)}$
- *shortfall* is defined as:
- $\sum_{a \in assets} P_a \times (C \cdot borrowed_a - supplied_a)$
- After an account loses a supply s_s of a given asset
  - $shortfall_{post} = shortfall_{pre} + P_s \cdot s_s$
  - This can be seen algebraically from above as:

    - $\sum_{a \in assets} P_a \times (C \cdot borrowed_a - supplied_a)$
    - $P_a \times (C \cdot B_a - S_a) + P_b \times (C \cdot B_b - S_b) + \ldots$
    - The term $P_s \times (C \cdot B_s - S_s)$ becomes $P_s \times (C \cdot B_s - (S_s - s_s))$
    - $P_s \times (C \cdot B_s - (S_s - s_s)) = P_s \times (C \cdot B_s - S_s) + P_s \cdot s_s$
    - This leaves us with $shortfall_{pre} + P_s \cdot s_s$
- Similarly, when an account loses a borrow b_b of a given asset:
    - $shortfall_{post} = shortfall_{pre} - C \cdot P_b \cdot b_b$
- After a liquidation, an account has lost **amountClose** of the borrowed asset in borrows, and **amountSeize** of the collateral asset in supply, and thus:
- $shortfall_{post} = shortfall_{pre} - P_B \cdot C \cdot amountClose + P_C \cdot amountSeize$
- Substituting **amountClose** and **amountSeize** from above:
- $shortfall_{post} = shortfall_{pre} - \frac{C \cdot P_B \cdot shortfall_{pre}}{P_B \cdot (C-L-1)} + \frac{P_C \cdot shortfall_{pre}(1+L)}{P_C \cdot (C-L-1)}$
- Cancelling matching P_B and P_C terms from fractions:
- $shortfall_{post} = shortfall_{pre} - \frac{C \cdot shortfall_{pre}}{(C-L-1)} + \frac{shortfall_{pre}(1+L)}{(C-L-1)}$
- Undistributing $shortfall_{pre}$
- $shortfall_{post} = shortfall_{pre} \times (1 - \frac{C}{(C-L-1)} + \frac{(1+L)}{(C-L-1)})$
- $shortfall_{post} = shortfall_{pre} \times (\frac{(C-L-1)}{(C-L-1)} - \frac{C}{(C-L-1)} + \frac{(1+L)}{(C-L-1)})$
- $shortfall_{post} = shortfall_{pre} \times (\frac{(C-L-1)-C+1+L}{(C-L-1)})$
- $shortfall_{post} = shortfall_{pre} \times (\frac{0}{C-L-1})$
- $shortfall_{post} = 0,\ and\ is\ undefined\ if\ C = L + 1$

Derivation of *discountedRepayToEvenAmount*

- $Given\ shortfall_{post} = shortfall_{pre} - \frac{C \cdot P_B \cdot shortfall_{pre}}{P_B \cdot x} + \frac{P_C \cdot shortfall_{pre}(1+L)}{P_C \cdot x}$
- $Find\ x\ such\ that\ shortfall_{post} = 0$
- Simplifying the equation above:
- $shortfall_{post} = shortfall_{pre} \times (1 - \frac{C}{x} + \frac{1+L}{x})$
- $shortfall_{post} = shortfall_{pre} \times (\frac{x-C+1+L}{x})$
- This is zero iff:
- $x - C + 1 + L = 0\ and\ x \neq 0,\ or:$
- $x = C - L - 1\ and\ C \neq L + 1$