

# M^0 Protocol Engineering Specification v1.1

M^0 Labs Engineering, March 2024

## Overview

M^0 is an EVM-compatible, immutable protocol that enables minting and burning of the ERC20 token \$M. It also allows for \$M distributions to yield earners and governance token (\$ZERO) holders. There are three main types of actors in the protocol - Minters, Validators, and Yield Earners - all of which are permissioned via governance. Protocol variables are also managed by governance and are stored in a Registrar configuration contract.

## Main Invariants

1.  $totalOwedM \geq totalMSupply$

### Reasonable parameter bounds:

30 days with no calls for *protocol.updateIndex*

MinterRate - up to 100% (10\_000 BPS APY)

EarnersBaseRate - up to 100% (10\_000 BPS APY)

2.  $totalOwedM = totalActiveOwedM + totalInactiveOwedM$

$$3. \text{totalMSupply} = \text{totalNonEarningMSupply} + \text{totalEarningMSupply}$$

## TTG-controlled variables

Name	Definition	Notes & Examples
<i>MintersList</i>	The list of Minters' addresses.	<b>TTG name:</b> <i>MINTERS_LIST</i>  [0xbCcA4494d525008f70Ba72Ac8D1A57B4D1908FcF, ..]
<i>ValidatorsList</i>	The list of Validators' addresses.	<b>TTG name:</b> <i>VALIDATORS_LIST</i>  [0xbCcA4494d525008f70Ba72Ac8D1A57B4D1908FcF, ..]
<i>EarnersList</i>	The list of whitelisted Yield Earner's addresses.	<b>TTG name:</b> <i>EARNERS_LIST</i>  [0xbCcA4494d525008f70Ba72Ac8D1A57B4D1908FcF, ..]

$mintRatio$  $MR$	<p>The ratio that defines the amount of <math>owedM_{minter}</math> a Minter may maintain relative to their  <math>(collateralValue_{minter} - totalRetrievalAmount)</math></p>	<p><b>TTG name:</b> <math>mint\_ratio</math></p> <p>There is \$9,000 of <math>CV</math> and the <math>MR</math> is 90%.</p> <p>The Minter may mint up to 8,100 \$M without incurring penalty charges.</p>
$minterInterestRateModel$  $minterRate$	<p>Yearly Interest rate that continuously accrues on \$M owed to the protocol.</p> <p>Smart contract that implements logic for calculation of <math>minterRate</math>.</p> <p>It implements <math>getRate()</math> method that returns value of <math>minterRate</math> in BPS.</p>	<p><b>TTG name:</b> <math>minter\_rate\_model</math></p> <p><b>APY % in BPS</b></p> <p><math>minterRate = 400</math> bps</p>
$earnerInterestRateModel$  $earnerRate$	<p>Yearly Interest rate that continuously accrues on \$M owned by earners.</p> <p>Smart contract that implements the logic for calculation of safe <math>earnerRate</math>.</p> <p>It implements <math>getRate()</math> method that returns value of <math>earnerRate</math> in BPS.</p>	<p><b>TTG name:</b> <math>earner\_rate\_model</math></p> <p><b>APY % in BPS</b></p> <p><math>earnerRate = 300</math> bps</p>
$penaltyRate$	<p>A discrete fee that is levied on:</p> <ol style="list-style-type: none"> <li>1. <math>owedM_{minter}</math> if <math>CV_{minter}</math> was not updated on time. Penalty is charged only once per every</li> </ol>	<p><b>TTG name:</b> <math>penalty\_rate</math></p> <p><b>penalty</b> = 0.1% in BPS</p> <p><math>penaltyRate = 10</math> bps</p>

	<p>missed <i>updateCollateralInterval</i> .</p> <p>2. <i>excessiveOwedM<sub>minter</sub></i> if it is present after the <i>CV<sub>minter</sub></i> was updated.</p>	
<i>mintDelay</i>	The amount of time that mint request is delayed by before it can be executed.	<p><b>TTG name:</b> <i>mint_delay</i></p> <p><i>mintDelay</i> = 14400 sec</p>
<i>mintTTL</i>	The amount of time that mint request can remain live before it can no longer be executed.	<p><b>TTG name:</b> <i>mint_ttl</i></p> <p><i>mintTTL</i> = 18000 sec</p>
<i>minterFreezeTime</i>	The amount of time that <i>Minter</i> stays frozen after one <i>freeze</i> call.	<p><b>TTG name:</b> <i>minter_freeze_time</i></p> <p><i>minterFreezeTime</i> = 86400 sec</p>
<i>updateCollateralInterval</i>	The length of time in seconds that <i>Minter</i> has to call <i>updateCollateral</i> , from the previous time it was called by that minter, before they will incur the penalty.	<p><b>TTG name:</b> <i>update_collateral_interval</i></p> <p><i>updateCollateralInterval</i> = 86400 sec</p>
<i>updateCollateralValidatorThreshold</i>	The number of validators required to verify the validity of <i>updateCollateral</i> data.	<p><b>TTG name:</b> <i>update_collateral_threshold</i></p> <p><i>updateCollateralThreshold</i> = 2</p>
<i>isEarnersListIgnored</i>	Reserved for future use to be able to drop any whitelisting checks for earners.	<p><b>TTG name:</b> <i>earners_list_ignored</i></p> <p><i>earnersListIgnored</i> = false</p>

## Protocol variables and definitions

Name	Definition	Notes & Example
$collateral_{minter}$  $CV_{minter}$	<p>The current value of t-bills in the Minter's SPV (Special Purpose Vehicle).</p> <p>Regularly verified by Validators:  <math>CV_{minter} = Eligible\ Collateral</math></p>	<p>There are \$10,000 of t-bills in the SPV.</p> <p>The <math>CV_{minter}</math> is \$10,000.</p>
$lastUpdate_{minter}$	<p>The timestamp of last collateral update by <i>Minter</i>.</p> <p>The minimum of all <i>Validators</i>' timestamps provided to verify the validity of collateral update.</p>	
$penalizedUntil_{minter}$	The timestamp of the end of the last penalization interval.	
$activeOwedM_{minter}$	The amount of \$M that <i>Minter</i> owes and has to repay to the protocol including accrued interest and penalties at any moment in time.	<p>Penalties are charged due to</p> <ul style="list-style-type: none"> <li>late collateral updates</li> <li>maintaining <math>excessiveActiveOwedM_{minter}</math></li> </ul>
$inactiveOwedM_{minter}$	The amount of \$M that deactivated <i>Minter</i> owes to the protocol.	

$maxAllowedActiveOwedM_{minter}$	<p>The maximum amount of \$M that is allowed to be owed to the protocol before it begins to accrue penalty charges after .</p> $maxAllowedActiveOwedM_{minter} = MCR * (CV_{minter} - totalActiveOwedM)$	<p>Mint requests are not accepted if this condition is not fulfilled.</p> <p>There is \$9,000 of <math>CV_{minter}</math> and the <math>MCR</math> is 90%. The <i>Minter</i> may mint up to 8,100 \$M without incurring penalty charges</p>
$totalActiveOwedM$	The total amount of <i>activeOwedM</i> for all <i>Minters</i> .	
$totalInactiveOwedM$	The total amount of <i>inactiveOwedM</i> for all <i>Minters</i>	
$pendingCollateralRetrieval_{minter, Id}$	Pending collateral retrievals for <i>Minter</i> with their ids and amounts.	
$totalPendingCollateralRetrievals_{minter}$	The total amount of all retrieval requests for <i>Minter</i>	

## Protocol variables used for interest accruals

Name	Definition	Notes & Example
<i>MinterGateway index</i>	The current value of global protocol index used for calculations of $activeOwedM_{minter}$	

	$MinterGateway\ index_{t1} = fn(index_{t0}, minterRate, t1 - t0)$	
<i>MinterGateway indexLatestUpdateTime</i>	The last timestamp when <i>index</i> was updated and stored.	
<i>MToken index</i>	<p>The current value of the global MToken index used for calculations of <math>balanceOf_{earner}</math></p> $MToken\ index_{t1} = fn(index_{t0}, earnerRate, t1 - t0)$	
<i>MToken indexLatestUpdateTimestamp</i>	The last timestamp when <i>MToken index</i> was updated and stored.	
$principalOfActiveOwedM_{minter}$	<p>The principal value of \$M adjusted to <math>index_t</math> where <math>t</math> is the timestamp of minting action:</p> $principalOfActiveOwed_{minter,t} += mintAmount / index_t$	
<i>totalPrincipalOfActiveOwedM</i>	The total value of <i>principalOfActiveOwedM</i> across all <i>Minters</i>	
$principalOfEarningSupply_{earner}$	The principal value of \$M that earns yield for <i>Earners</i> .	
<i>totalPrincipalOfEarningSupply</i>	The total principal value of \$M that earns yield for all <i>Earners</i>	

## Protocol and M Token actors and actions

Actors	Allowed actions
<i>Minter</i>	<ul style="list-style-type: none"><li>• <i>updateCollateral</i></li><li>• <i>proposeMint</i></li><li>• <i>mintM</i></li><li>• <i>burnM</i></li><li>• <i>proposeRetrieval</i></li></ul>
<i>Validator</i>	<ul style="list-style-type: none"><li>• <i>freezeMinter</i></li><li>• <i>cancelMint</i></li></ul>
<i>Earnner</i>	<ul style="list-style-type: none"><li>• <i>startEarning</i></li><li>• <i>stopEarning</i></li></ul>
<i>Anyone</i>	<ul style="list-style-type: none"><li>• <i>activateMinter</i></li><li>• <i>burnM [for Minter]</i></li><li>• <i>deactivateMinter</i></li><li>• <i>updateIndex</i></li></ul>



## Core functions

*updateCollateral(collateral, retrievalIds[], metadata, validators[], timestamps[], signatures[]): minimumTimestamp*

- **Closes** all pending *retrievalIds[]* requests
- **Updates**  $CV_{minter}$
- **Accrues**  $penalty_{minter}$  for missed update collateral intervals and/or  $excessiveOwedM_{minter}$  if present after  $CV_{minter}$  was updated
- **Called** at least once per *updateCollateralInterval* by *Minter* or the entire  $activeOwedM_{minter}$  imposes the *penalty*.
  - **Calculate** *updateCollateral* message digest for each validator *signature* .
  - **Verify** that each signature is valid, unique, and is not in the future, and is coming from TTG-approved *Validator*
  - **Verify** that at least *updateCollateralThreshold* of valid unique signatures of TTG-approved *Validators* were Provided and **Find** the minimum of given *Validators timestamps[]*.
  - **Resolve** pending retrievals. Deduct the amount of retrieval from  $totalPendingCollateralRetrievals_{minter}$  for every retrieval. **Delete** *retrievalId* from  $pendingCollateralRetrievals_{minter}$   
$$totalPendingCollateralRetrievals_{minter} -= \sum_i retrieval_{minter, retrievalId[i]}$$

- **Re-calculate** current  $activeOwedM_{minter}$ , and **Impose**  $penalty_{minter}$  for every missed and not yet penalized update collateral interval. Update  $penalizedUntilTimestamps_{minter}$  if a penalty was imposed.

$$penalty_{minter} = activeOwedM_{minter} * missedIntervalsNum * penalty \text{ where}$$

$$missedIntervalsNum = now - \max(lastCollateralUpdate_{minter}, lastPenalizedUntil_{minter}) / updateCollateralInterval$$

$$penalizedUntilTimestamps_{minter} = \max(lastCollateralUpdate_{minter}, lastPenalizedUntil_{minter}) + missedIntervalsNum * updateCollateralInterval$$

- **Add** penalty to  $principalOfActiveOwedM_{minter}$  and  $totalPrincipalOfActiveOwedM$   
 $principalOfActiveOwedM_{minter} += penalty_{minter} / index_{now}$ ,

$$totalPrincipalOfActiveM += penalty_{minter},$$

$$\text{After penalization new } activeOwedM_{minter, t1} = activeOwedM_{minter, t0} + penalty_{minter}$$

- **Verify**  $minimum\ of\ timestamps[]$  is greater than previous collateral update's  $timestamp_{minter}$ .

- **Update**  $CV_{minter}$

$$CV_{minter} = collateral$$

- **Re-calculate** current  $activeOwedM_{minter}$  and **Impose**  $penalty_{minter}$  for  $excessiveActiveOwedM_{minter}$  if it is present after setting new  $collateral$  and resolving  $retrievalIds[]$

$$penalty_{minter} = excessiveActiveOwedM_{minter} * penalty, \text{ where}$$

$$excessiveActiveOwedM_{minter} = activeOwedM_{minter} - maxAllowedActiveOwedM_{minter}$$

- **Call**  $updateIndex$ .

- **Returns** *minimum of timestamps*[]

*proposeMint(amount, destination): mintId*

- **Proposes** new \$M mint and returns id of pending mint proposal.
- **Called** by *Minter* approved by TTG who is not frozen at the moment.
- **Specifies** the amount of \$M to mint and the *destination* address to which they'd like it to be minted.
- Any *Validator* can cancel this transaction within *mintDelay* + *mintTTL* until it is executed.
- Only one active proposal per *Minter*. New mint proposal replaces the existing one.
  - **Re-calculate** current *activeOwedM<sub>minter</sub>*
  - **Check** that mint request is sufficiently collateralized:  

$$(CV_{minter} - totalPendingCollateralRetrievals_{minter}) * MR \geq owedM_{minter} + amount,$$

NOTE: If the  $CV_{minter}$  was not updated on time, the protocol assumes that it is set to 0.
  - **Create** pending for *mintDelay* time Mint proposal
  - **Return** unique *mintId* of Mint proposal.

*mintM(mintId)*

- **Executes** the *mintId* proposed mint created by *proposeMint()*.
  - **Check** that *mintId* request delay time is over and it is not expired yet

- **Check** that *Minter* is still approved by TTG and is not frozen.
- **Re-calculate** current  $owedM_{minter}$
- **Check** that minter is still sufficiently collateralized :  

$$(CV_{minter} - totalPendingCollateralRetrievals_{minter}) * MCR \geq owedM_{minter} + amount$$

NOTE: If the  $CV_{minter}$  was not updated on time, the protocol assumes that it is set to 0.
- **Delete** *mintId* proposal
- **Update** *totalPrincipalOfActiveOwedM* and *principalOfActiveOwedM<sub>minter</sub>*  

$$totalPrincipalOfActiveOwedM += amount / index_{now}$$

$$principalOfActiveOwedM_{minter} += amount / index_{now}$$
- **Mint** \$M to *destination* address.
- **Call** *updateIndex*.

*cancelMint(minter, mintId)*

- **Cancels** valid pending *mintId* proposal. **Deletes** *mintId* proposal.
- **Can be called** any time until mint request is executed by *Minter*: within a maximum of  $mintDelay + mintTtl$ .
- **Can be called** by any *Validator* approved by TTG.

*freezeMinter(minter)*

- **Called** by any *Validator* to temporarily freeze *proposeMint* and *mintM* for *freezeTime* on a specific *minter*.
- **Can be called** at any time again during the frozen period to extend the frozen period by an additional *freezeTime*.  
 $frozenUntil = block.timestamp + minterFreezeTime$
- **Does not** freeze any functions besides *proposeMint* and *mint*.

*burnM(minter, maxPrincipalAmount, maxAmount)*

- **Decreases**  $owedM_{minter}$  by *amount*. \$M is subtracted from the caller of this function.
  - **Re-calculate** current  $activeOwedM_{minter}$  and impose *penalty* for missed update collateral intervals if applicable.
  - **Reduce**  $activeOwedM_{minter}$  by  $\min(amount, activeOwedM_{minter})$  if *Minter* is active  
 $totalPrincipalOfActiveOwedM \ -= \ \min(amount, activeOwedM_{minter}) / index_{now}$   
 $principalOfActiveOwedM_{minter} \ -= \ \min(amount, activeOwedM_{minter}) / index_{now}$

**OR**

**Reduce**  $inactiveOwedM_{minter}$  by  $\min(amount, inactiveOwedM_{minter})$  if *Minter* was deactivated

$totalInactiveOwedM \ -= \ \min(amount, inactiveOwedM_{minter})$

$inactiveOwedM_{minter} \ -= \ \min(amount, inactiveOwedM_{minter})$

- **Burn**  
 $\min(amount, activeOwedM_{minter})$  or  $\min(amount, inactiveOwedM_{minter})$  of \$M from the caller of the function, e.g.  
*msg.sender*.

- **Call** *updateIndex*.

*burnM(minter, maxAmount)*

- **Decreases**  $owedM_{minter}$  by *amount*. \$M is subtracted from the caller of this function.
- **Calculates** *maxPrincipalAmount* to burn given current value of MinterGateway index
- **Call** *burnM(minter, maxPrincipalAmount, maxAmount)*

*proposeRetrieval(amount): retrievalId*

- **Allows** *Minter* to retrieve t-bills from the SPV. Called only by *Minter*
  - **Re-calculate** current *activeOwedM*.
  - **Check** that the retrieval proposal does not put Minter into under collateralization zone

$$activeOwedM_{minter} \leq (CV_{minter} - totalPendingCollateralRetrievals_{minter} - amount) * MR$$

- **Check** that the retrieval proposal and current retrieval proposal won't exceed collateral
 
$$totalPendingCollateralRetrievals_{minter} + amount < CV$$
- **Add** *amount* to  $totalRetrieveAmount_{minter}$ .
 
$$totalPendingCollateralRetrievals_{minter} += amount$$
- **Save**  $pendingRetrievals_{minter}$ 

$$pendingRetrievals_{minter, retrievalId} = amount$$

- **Return** unique *retrievalId* for a retrieval proposal.

*activateMinter(minter)*

- **Demonstrates** that *minter* has been added to the *Minters* list by TTG governance.
  - **Check** that *minter* is in *Minters* TTG list and has not been activated yet.
  - **Set**  $isActive_{minter} = true$ .

*deactivateMinter(minter)*

- **Demonstrates** that *minter* has been removed from the *Minters* list by TTG governance.
- **Required** to adjust *totalActiveOwedM* and *totalInactiveOwedM*.
  - **Check** that *minter* is not in *Minters* list but has  $activeOwedM_{minter}$  accruing interest.
  - **Subtract** *minter's*  $activeOwedM_{minter, now}$  from *totalActiveOwedM* and add it to *totalInactiveOwedM*  
 $inactiveOwedM_{minter} = activeOwedM_{minter} + penalty_{minter}$  for missed collateral updates if any;  
 $totalActiveOwedM -= inactiveOwedM_{minter}$   
 $totalInactiveOwedM += inactiveOwedM_{minter}$ .
  - **Set**  $isActive_{minter} = false$ .
  - **Set**  $isDeactivated_{minter} = false$ .
  - **Reset** *minter's* protocol state.
  - **Call** *updateIndex*.

---

## M Token

---

### *startEarning()*

- **Allows** a user to start earning yield on their \$M balance.  $balanceOf_{user}$  will be rewritten to store  $principalOfEarningSupply_{MToken, earner}$ .
  - **Check** that  $msg.sender$  is on the Approved by TTG Earners List.
  - **Save**  $balanceOf_{MToken, earner} = principalOfEarningSupply_{MToken, earner} = balanceOf_{MToken, earner} / index_{now}$
  - **Save**  $totalPrincipalOfEarningSupply += balanceOf_{M, earner} / index_{now}$
  - **Save**  $totalNonEarningSupply -= balanceOf_{MToken, earner}$

### *stopEarning()*

- **Allows** a user to stop earning yield on their \$M balance.  $balanceOf_{user}$  after the call of this function will be rewritten to store  $balanceOf_{now, MToken, earner} = principalOfEarningSupply_{MToken, earner} * index_{now}$ .
  - **Save**  $balanceOf_{MToken, earner} = principalOfEarningSupply_{MToken, earner} * index_{now}$
  - **Save**  $totalNonEarningSupply += balanceOf_{MToken, earner}$



- **Save**  $totalPrincipalOfEarningSupply \ -= \ balanceOf_{MToken, earner} / index_{now}$

*transfer(sender, recipient, amount)*

- **Allows** a user to transfer \$M between both - yield earning and non-yield earning participants.

- **Deduct** *amount* from *sender* and **Add** to *recipient*
- 

- **If** *sender* and *recipient* are both yield earning or non-yield earning participants

$$balanceOf_{MToken, sender} \ -= \ amount$$

$$balanceOf_{MToken, recipient} \ += \ amount$$

- **If** *sender* is yield earning and *recipient* is non-yield earning participant

$$balanceOf_{MToken, sender} \ -= \ principalOfEarningSupply_{MToken, sender} = amount / index_{now}$$

$$totalPrincipalOfEarningSupply \ -= \ amount / index_{now}$$

$$balanceOf_{MToken, recipient} \ += \ amount$$

$$totalNonEarningSupply \ += \ amount$$

- **If** *sender* is non-yield earning and *recipient* is yield earning participant

$$balanceOf_{MToken, sender} \ -= \ amount$$

$$totalNonEarningSupply \ -= \ amount$$

$$balanceOf_{MToken, recipient} \ += \ principalOfEarningSupply_{MToken, recipient} = amount / index_{now}$$

$$totalPrincipalOfEarningSupply \ += \ amount / index_{now}$$

- **Call** *updateIndex*.

---

## Interest Calculations and Indices

---

*MinterGateway.updateIndex()*

- **Calculates and updates** *minterIndex* and *earnerIndex* to their current values.
  - **Distribute** Excess \$M to ZERO Vault holders  
 $excessM = totalOwedM - totalSupply\ of\ M\ Token, \text{ if } totalOwedM > totalSupply\ of\ M$
  - **Calculate and update** *minterIndex* :  
 $minterIndex_{now} = fn(minterIndex_{minterIndexLastUpdatedTimestamp}, rate, now - minterIndexLastUpdatedTimestamp)$
  - **Set** *latestMinterRate* = *minterInterestRateModel.getRate()*
  - **Calculate and update** *earnerIndex* :  
 $earnerIndex_{now} = fn(earnerIndex_{earnerIndexLastUpdatedTimestamp}, rate, now - earnerIndexLastUpdatedTimestamp)$
  - **Set** *latestEarnerRate* = *earnerInterestRateModel.getRate()*
  - **Set** *minterIndexLastUpdatedTimestamp* = *now*
  - **Set** *earnerIndexLastUpdatedTimestamp* = *now*.

**NOTE:**  $index2 = index1 * e(rate * timeElapsed)$ , using Pade(4, 4) approximation.

$timeElapsed = now - indexLastUpdatedTimestamp,$

$$e(x) = (1 + x/2 + 3(x^2)/28 + x^3/84 + x^4/1680) / (1 - x/2 + 3(x^2)/28 - x^3/84 + x^4/1680)$$

*MinterGateway.currentIndex()*

- **Calculates and returns** current value of *minterIndex*.

$$minterIndex_{now} = fn(minterIndex_{minterIndexLastUpdatedTimestamp}, now - minterIndexLastUpdatedTimestamp)$$

**E.g.**  $minterIndex_{now} = minterIndex_{minterIndexLastUpdatedTimestamp} * e^{minterRate * (now - minterIndexLastUpdatedTimestamp)}$

*MToken.updateIndex()*

- **Calculates and updates** *earnerIndex* to their current values.

- **Calculate and update** *earnerIndex* :

$$earnerIndex_{now} = fn(earnerIndex_{earnerIndexLastUpdatedTimestamp}, rate, now - earnerIndexLastUpdatedTimestamp)$$

- **Set** *latestRate* = *earnerInterestRateModel.getRate()*
- **Set** *earnerIndexLastUpdatedTimestamp* = *now*.

*MToken.currentIndex()*

- **Calculates and returns** current value of *earnerIndex*.

$$earnerIndex_{now} = fn(earnerIndex_{earnerIndexLastUpdatedTimestamp}, now - earnerIndexLastUpdatedTimestamp)$$

**E.g.**  $earnerIndex_{now} = earnerIndex_{earnerIndexLastUpdatedTimestamp} * e^{earnerRate * (now - earnerIndexLastUpdatedTimestamp)}$

*activeOwedMOF(minter)*

- **Calculates and returns the** current value of owed \$M per *minter*.
  - **Calculate** *minterIndex* :  

$$minterIndex_{now} = fn(minterIndex_{minterIndexLastUpdatedTimestamp}, now - minterIndexLastUpdatedTimestamp)$$
  - **Returns:**  $principalOfActiveOwedM_{minter} * minterIndex_{now}$

---

## Interest Rate Models

---

Stable Earner Interest Rate model contract

*earnerRateModel.rate()*

- **Returns** current earner rate in basis points.

$$P1 * e^{r1 * t} - P1 = P2 * e^{r2 * t} - P2$$

- **If**  $totalActiveOwedM \leq totalEarningSupply$ ,  
 $r2 = minterRate * totalActiveOwedM / totalEarningSupply$ ,
- **Otherwise, calculate** equilibrium rate  

$$P1 * e^{r1 * t} - P1 = P2 * e^{r2 * t} - P2$$

$$r2 = \ln(1 + (P1 * e^{r1 * t} - P1) / P2),$$

$P1 = totalActiveOwedM, P2 = totalEarningSupply, t = confidence\ interval$

- **Return**  $\min(baseEarnerRate, MULTIPLIER * r2),$

where *baseEarnerRate* and *minterRate* are APY in basis points.

Minter Interest Rate model contract

*minterRateModel.rate()*

- **Returns** current constant *minterRate* in basis points.