Wrapped M Engineering Spec V0.3

Non-rebasing ERC20 token equivalent of M

M^0 Labs Engineering, July 2024

Overview

Raw **M** is a rebasing ERC20 yield-accruing token. **`balanceOf`** and **`totalSupply`** of M are growing in value as the earning rate continues to accrue for earners whitelisted by TTG governance.

Even though this approach leads to minimalist efficient yield distribution, it can break certain Defi integrations. Additionally, if earners supply M to Defi protocols, they cannot continue to accrue interest on their M balance.

Challenges

- Not all Defi protocols support rebasing tokens
  Note: Rebasing tokens - balance change/accruing of yield happens without user interaction.

- Earners would like to continue accruing yield while M is used in Defi and not kept in their wallets

- Non-earners should not receive the earner's yield while unwrapping Wrapped M.

Goals

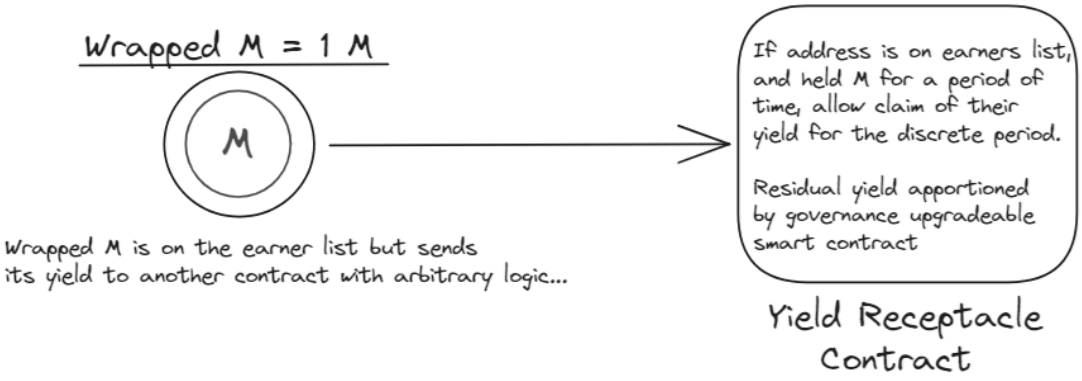
- ERC20 non-rebasing version of token used in Defi ecosystem.

- Accrued yield is accessible via the claim function.

- Earners preserve the possibility to earn even if they don't hold M.

- Additional overriding functionality allows the yield of certain earners to be claimed by another address.

Product requirements



Tech Design Assumptions

- Wrapped M will be added to the TTG earners list;
- M will be wrapped and unwrapped into Wrapped M; M wrapped supply will be locked in the wrapper and will continue to accrue interest;
- All raw M locked in Wrapped M will be earning yield;
- Shared TTG earners list between M and Wrapped M;
- Wrapped M token will use the M token index for yield accrual;
- Existence of M excess yield in Wrapped M.
  Note: All M locked in Wrapped M will accrue yield, but not all Wrapped M will necessarily accrue yield at any moment. There will be an excess of M yield.
- Excess of M will be sent to the Distribution Vault.

Invariants

$$M\ Total\ Balance\ locked\ in\ Wrapped\ M\ +\ M\ Total\ Accrued\ Yield\ >=\ Wrapped\ M\ Total\ Supply\ +\ W$$
$$M\ Yield\ excess\ in\ Wrapped\ M\ =\ M\ Total\ Balance\ locked\ in\ Wrapped\ M\ +\ M\ Total\ Accrued\ Yield\ -$$

M vs Wrapped M

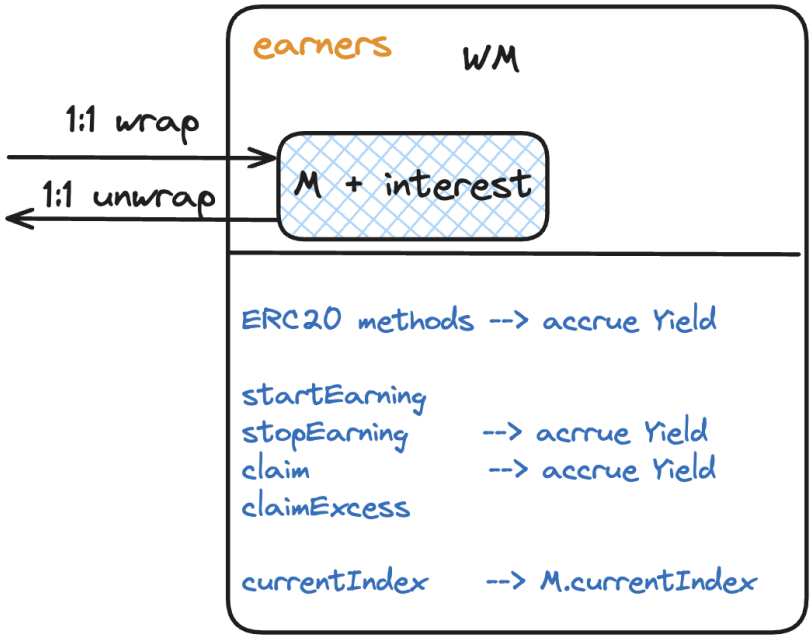


Diagram showing the flow of funds as well as the methods that will accrue yield when called

	M	**Wrapped M (wrapper)**
ERC20	YES	YES
Rebasing	YES	NO
Earner balanceOf	$M\ principal\ *\ current\ index$	$Wrapped\ M\ principal\ *\ index\ at\ last\ account\ action$
Non-earner balanceOf	$M\ balance$	$Wrapped\ M\ balance$
Earning totalSupply	$M\ principal\ *\ current\ index$	$Wrapped\ M\ principal\ *\ last\ index$
Non-earning totalSupply	$M\ total\ supply$	$Wrapped\ M\ total\ supply$

totalSupply	A sum of total supplies	A sum of total supplies
Accrued yield per account calculation	None	$Wraped\ M\ principal\ *\ (current\ index\ -\ index\ at\ last\ account\ action)$
Accrued total supply yield calculation	None	$Wrapped\ M\ principal\ *\ (current\ index\ -\ last\ index)$
Additional functionality	None	• wrap/unwrap • claim • claimExcess

Wrapped M token interface

Wrapper specific methods

wrap(destination, amount)

- Transfer amount of M tokens from the caller to the Wrapped M contract.
- Mint amount of Wrapped M tokens to *destination* address.
- If the *destination* address is on the TTG earners list, any accrued yield for the *destination* address is realized/minted, and the current M principal supplied and M index are saved for the *destination* address.
- Adjust total earning supply and total non-earning supply respectively for the *destination* address being an earner or non-earner.

unwrap(destination, amount)

- Transfer amount of M to the *destination* address.
- Burn amount of Wrapped M from the caller.
- If the *caller* address is on the TTG earners list, any accrued yield for the *caller* address is realized/minted, and the current M principal supplied and M index are saved for the *caller* address.
- Adjust total earning supply and total non-earning supply respectively for the *caller* address being an earner or non-earner.

enableEarning()

- Call M.startEarning for Wrapped M address
- Revert if earning was already enabled in the past.
  Note: The current wrapper allows to enable/disable earning only once.

disableEarning()

- Call M.stopEarning for Wrapped M address
- Save M index when earning was disabled

startEarningFor(account)

- Revert if *account* is not on the TTG earners list.
- Revert if earning was not enabled.
- Accrue yield for *account*.
- Convert *account* balance from raw balance format to principal and index format

stopEarningFor(account)

- Revert if *account* is on the TTG earners list.
- Accrue yield for *account*.
- Convert *account* balance from principal and index format into raw balance format.

claimFor(account)

- Accrue yield if the account is an earner.
- Transfer yield to another account if the overriding feature is set.

claimExcess

- Send excess of M yield into the Distribution Vault.

currentIndex

- Returns current M index or M index when earning was disabled.

totalNonEarningSupply

- Returns the total supply held by all non-earning accounts.

totalEarningSupply

- Returns the non-rebasing total supply held by all earning accounts.

accruedYieldOf(account)

- Returns the accrued (unrealized) yield for an earning account since its last claim.

totalAccruedYield

- Returns the accrued (unrealized) yield for all earning accounts since their last claims.

ERC20-like standard methods

transferFrom(sender, recipient, amount)

- Accrue yield for recipient.
- Accrue yield for sender (caller).
- Transfer amount of Wrapped M tokens from sender to recipient.

transfer(recipient, amount)

- Same as above, assuming caller is sender.

approve(spender, amount)

- Allow a spender to spend some amount on behalf of an account.

balanceOf(account)

- Returns the non-rebasing balance of an account.

allowance(spender, account)

- Returns the amount a spender can spend on behalf of an account.

totalSupply

- Returns the non-rebasing total supply of all accounts.

Migrator upgradability pattern

Wrapped M token uses the Migrator pattern introduced by Maple.

Each upgrade path from one implementation to another will be facilitated by a delegatecall to a migrator contract whose sole purpose (via a fallback function) is to perform checks, manipulate storage, and set the implementation slot.

The rationale for this pattern is:

- Reduce potential bloat in new implementation by extracting and isolate all unique one-time use upgrade logic into its own contract
- Group all upgrade, initialization, checks, and upgrade logic into a single auditable atomic contract/function
- Allow for complex migrations in a more readable format, since there can be steps that can occur before and after the implementation slot is set
- Reduce the need for passing any upgrade/migrate arguments as they can all be hardcoded or derived in the migrator

Each upgradeable Wrapped M contract will have **`migrate`** method, that anyone can call and that takes no arguments, that will facilitate migration and upgrade by executing the following steps:

1. read a specific/unique TTG Registrar value to get the address of the migrator
2. optionally execute some pre-upgrade ‘migration’ logic
3. upgrade implementation
4. optionally execute some post-upgrade ‘migration’ logic

Eventually, Wrapped M can be upgraded to the version without the **`migrate`** method, preventing the possibility of further upgrades.

Example of Migrator contract

Wrapped M Token implementations are expected to have a unique migrator prefix version in each subsequent implementation. Example `_MIGRATOR_V1_PREFIX` for the first version. This ensures that there is a unique migrator key at TTG Registrar for each version.

The first version of the Wrapped M Token implementation will also have another **`migrate`** method, that only an admin can call and that takes a migrator as an argument. This is to allow a more centralized upgrade path by bypassing TTG, in the earlier versions. This method can be dropped once this centralized upgrade path is deemed no longer necessary.