

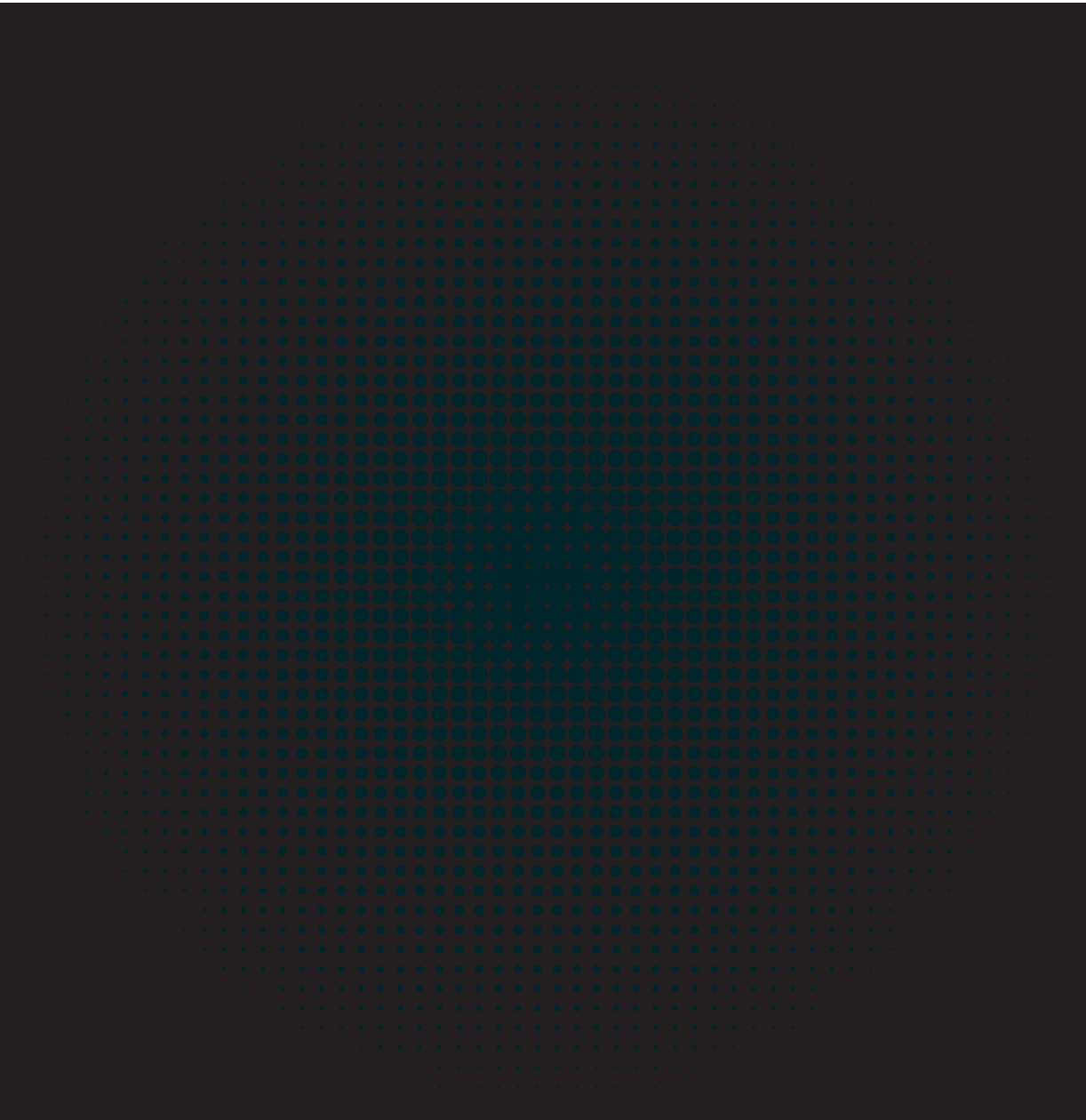
cmta.

CMTAT Framework

**Blockchain agnostic functional specifications of the
CMTA Token for the tokenization of financial instruments.**

First published January 2022

Updated November 2024, September 2025 and June 2026



CMTAT Framework: Blockchain agnostic functional specifications of the CMTA Token for the tokenization of financial instruments.

Capital Markets and Technology Association
Route de Chêne 30
1208 Genève

First Published: January 2022
Updated: November 2024
September 2025
June 2026

admin@cmta.ch
+41 22 73 00 00

No modification or translation of this publication may be made without prior permission. Applications for such permission, for all or part of this publication, should be made to the CMTA Secretariat by email to:

admin@cmta.ch

Table of Contents

1.	INTRODUCTION	03
2.	OVERVIEW AND KEY PRINCIPLES OF THE CMTAT FRAMEWORK	03
3.	OUTLINE OF THE CMTAT FRAMEWORK: KEY FUNCTIONALITIES	06
4.	REFERENCE IMPLEMENTATIONS	12
5.	CONCLUSION	14

1. INTRODUCTION

The CMTA Standard Token for Securities (CMTAT) is an open standard for smart contracts designed specifically for the tokenization of financial instruments. It is a framework that defines necessary and optional functions that can be used for tokenizing financial instruments such as equity, debt and structured products and other transferable securities. It is blockchain agnostic in that it defines a set of functionalities that a security token should implement. Reference implementations exist for Ethereum and Tezos showing how the CMTAT can be implemented (<https://github.com/CMTA/>).

The CMTAT framework is published by the Capital Markets and Technology Association (CMTA) and has been carefully designed and written by a working group including technology companies, law firms and financial institutions, by first defining the requirements for tokenizing financial instruments and translating those requirements into the list of CMTAT functionalities.

2. OVERVIEW AND KEY PRINCIPLES OF THE CMTAT FRAMEWORK

§ 2.1 *Tokenization and smart contracts*

Tokenization is a process that involves associating a financial instrument (such as a security), issued by a public or private issuer, with a digital token recorded on a distributed ledger such as a blockchain. Consequently, the tokenization process assumes that an issuer can not only issue financial instruments in a legally valid manner, but also create tokens that “behave” like the associated securities. For example:

- Capability to mint new tokens when new financial instruments are issued.
- Capability to burn existing tokens when the financial instruments are cancelled.
- Capability to transfer tokens when the associated financial instruments are transferred.
- Legal title over the financial instrument must follow control over the associated token, so that one cannot be transferred without the other.
- The functionalities for the transfer of a digital token must be aligned with the terms of the associated financial instrument for what regards transfer of title (e.g. for what regards potential transfer restrictions).
- Other than under the circumstances contemplated by applicable law (e.g. inheritance), tokens must not remain behind on the ledger when the legal title of the associated financial instrument is transferred, and conversely the legal title to the tokenized instrument must not remain behind when the token is transferred on the ledger.

As a consequence, tokenization requires tokens to have certain properties that can be adjusted to the specificities of the associated financial instruments. The CMTAT framework makes it possible to generate such tokens.

The issuer may want the ability to perform a burn and mint atomically within the same transaction. On some blockchains, such as Solana, this functionality is supported natively by the protocol, which allows multiple operations to be batched together in a single transaction. On Ethereum, however, this capability is not available by default. In such cases, it can be implemented either through a `multicall` function in the smart contract, which batches operations, or via a dedicated `burnAndMint` function.

§ 2.2 Eligible financial instruments

The CMTAT can be used for the tokenization of any financial instrument that can be associated with a digital token recorded on a distributed ledger so that the instrument cannot be transferred without the token, and vice versa. For this to be possible, the instrument must be generally transferable and the transfer of legal title to the instrument must be possible by transferring a digital token on a distributed ledger. Therefore, financial instruments whose transfer are subject to specific forms (e.g. wet ink assignments) or registration in a central registry (e.g. a land registry) are therefore not suitable for tokenization. However, the existence of statutory or contractual transfer restrictions do not automatically render the relevant instrument ineligible for tokenization within this framework. Design choices can be made to implement transfer restrictions and ensure that the movement on the ledger mirrors the transfer of the title to the associated financial instrument.

The CMTAT is primarily designed for the tokenization of “securities”, in the sense of fungible and transferable financial instruments. However, it can also be used for the tokenization of financial instruments that do not qualify as securities, such as non-fungible derivatives.

Whether a particular financial instrument is suitable for tokenization depends on the law applicable to the issuer and the relevant instrument, as well as the constitutive documents of the issuer. The CMTA publishes tokenization models for the tokenization of securities issued by Swiss issuers under Swiss law, but the CMTAT can also be used for the tokenization of securities of other issuers and financial instruments governed by other laws. Issuers must assess on a case-by-case basis whether the CMTAT is suitable for the tokenization of such securities.

§ 2.3 Standardization

Various methods exist for creating tokens suitable for tokenization and many smart contracts are used for this purpose in practice. Using standard frameworks has practical advantages, because it reduces the need for users to spend time and energy understanding the code and assessing the smart contract’s robustness and appropriateness. Reducing complexity lowers the costs of tokenization, while increasing predictability and the trust of market participants in the robustness of the tokenization process.

§ 2.4 Open source character of the code and security audits

The CMTAT reference implementation and its engines are released as open source code, granted under Mozilla Public License 2.0 (MPL) or MIT License. Those licenses allow developers to use, modify, and share code with the condition, for what regards the MPL, that any changes to licensed files are shared under the same license, while enabling combination with proprietary code in other files.

The open source character of the CMTAT’s reference implementation and the related engines guarantees that the code is subject to wide scrutiny, and makes it possible for the code to improve over time.

CMTA’s policy is to submit the CMTAT reference implementations that it supports to security audits, and make the resulting reports publicly available. However, there may be a delay between the release of a particular CMTAT implementation and the publication of the corresponding audit report. Users are responsible for checking whether any particular implementation of the CMTAT has been audited before using or relying on such implementation. As open source code, the CMTAT’s implementations are provided on an “as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that such code is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the CMTAT code is

exclusively with the user thereof. Should any part of any implementation of the CMTAT or its engines prove defective in any respect, the relevant user (not the CMTA or any contributor) will assume the cost of any necessary servicing, repair, or correction.

The CMTAT framework and its reference implementations have been developed by various persons. The reference implementations of the CMTAT are published by the CMTA and made available as open source code with the authorization of the copyright owners. CMTA and its members have not claimed any patent rights on the CMTAT, nor are they aware of third parties having claimed such rights.

§ 2.5 Chain agnosticism

The CMTAT framework is blockchain-agnostic, meaning it defines necessary and optional functionalities for tokens used in the tokenization of securities, without being linked to a specific technology. The CMTAT's reference implementation is in the Solidity language (suitable for Ethereum, Polygon, and other EVM-based chains). Other implementations currently exist for the Tezos and Aztec networks, as well as specifications for implementing the CMTAT framework on Solana. Other implementations are possible and encouraged.

CMTA considers chain agnosticism a key pillar in the development of blockchain-based infrastructures, as it allows chains to evolve and improve over time, and avoids tying key market infrastructures to specific ledgers. CMTAT's chain agnosticism also permits the use of both public and permissioned ledgers.

The present document outlines the necessary and optional functionalities that tokens must have to be used for tokenizing financial instruments, in a chain-agnostic way. Further information on the functionalities that are specific to each technological implementation can be found in the relevant CMTAT GitHub repositories (see Section 3 below).

§ 2.6 Modular character

The CMTAT is a modular framework that supports various types of tokenization. The framework includes "mandatory functions," which are necessary for any tokenization process. When CMTAT's mandatory functions are solely applied, the tokens primarily serve as "pointers," providing evidence of legal ownership of the associated financial instruments. Corporate actions on the relevant instruments, such as dividend or interest payments, must then be carried out "off chain," typically through bank transfers. However, additional optional functionalities can be used to execute corporate actions on-chain.

§ 2.7 Governance layer

One of CMTAT's defining features is its governance layer, which issuers can integrate with the basic token structure to ensure compliance with applicable laws (such as anti-money laundering laws or international sanctions) or the special terms of specific instruments (such as transfer restrictions) using the Validation Module's functions. In the CMTAT framework, governance is defined at the token level – and is consequently controlled by the issuer – and not at the level of any issuance or trading platform.

The Validation Module's functions are allocated to the issuer of the tokenized instrument by default, with the result that issuers are not dependent on third parties to ensure compliance with applicable laws and regulations. Issuers can however choose to delegate governance function management to a third party, such as a regulated financial intermediary or trading platform. This avoids the need for issuers to manage compliance functions themselves.

3. OUTLINE OF THE CMTAT FRAMEWORK: KEY FUNCTIONALITIES

The CMTAT framework defines the functions necessary to (i) create tokens that are suitable for association with financial instruments for tokenization purposes, and (ii) manage certain events affecting the relevant financial instruments using on-chain or off-chain mechanisms, as appropriate.

The framework also defines the “roles” of the various actors involved in the issuance and trading of tokenized securities, namely the issuer, tokenholders, and third parties. The framework distinguishes between “issuer functionalities” and “user functionalities” and the symbols below are used to identify roles in the CMTAT framework :

Role	Description
 Issuer	functionalities that can be activated (or “called”) by the issuer or another person designated by the issuer
 User	functionalities that can be “called” by the holders of the private keys associated with the ledger addresses on which the relevant tokens are recorded.

As mentioned above, the CMTAT distinguishes between mandatory functionalities and optional functionalities, which are grouped into “modules”.

The list of functionalities described below represents the minimal set of functionalities that must be implemented with respect to each module. Additional platform-specific functionalities may be required for compliance with certain platform-specific application programming interfaces (APIs).

It is recommended to use a token based on the ERC-20 standard for projects on Ethereum and EVM blockchains.

§ 3.1 *Mandatory functionalities*

The mandatory functions that a token should have to represent a financial instrument under the CMTAT framework are:

3.1.1 Base module mandatory functions

CMTAT’s base module contains the basic functionalities a token must have to be associated with an equity security or any other form of security. The base module provides the functionality that is essential for a CMTAT token to be a fungible token circulating on a blockchain.

Functionalities:

1. **Know total supply:** For a particular CMTAT token, any person may know the total number of tokens in circulation at any point in time. 
2. **Know balance** Each token holder may know the number of tokens currently recorded on their address. The issuer or any third-party appointed by the issuer may also know the number of tokens recorded on an address. 
3. **Transfer tokens:** Users may transfer some or all of their tokens to another ledger address (that the transferor does not necessarily control). 
4. **Create tokens:** Issue a given number of tokens to a given ledger address. 
The issuer must be in a position to create new tokens by allocating them to a distributed ledger address. This function is meant to be used when the issuer tokenizes newly issued securities (e.g. capital increase or re-opening of a particular debt issuance) or existing securities previously issued in a different form (e.g. in the form of paper certificates).
5. **Cancel tokens:** Cancel a given number of tokens from a given ledger address. 
The functionality can be used to cancel tokens in case of capital decrease or cancellation of debt instruments, if the issuer decides to retire tokens to have its securities issued in a non-tokenized format, or to comply with court orders requiring the cancellation of specific tokens).
6. **Pause tokens:** Prevent all transfers of tokens on the ledger until “Unpause” is called. 
The functionality allows the issuer to “pause” transfers to prevent the completion of transactions on the ledger. It can be used to block transactions in case of “hard forks”, pending a decision of the issuer as to which version of the ledger it will support. It is up to the issuer to decide whether token creation and deletion operations are also affected by the pause.
7. **Unpause tokens:** Restore the ability to transfer tokens on the ledger, in principle when all transfers are paused. 
8. **Know pause status:** For a particular CMTAT token, any person may know whether the contract is paused. 
9. **Deactivate contract:** This function can be called if it is necessary to permanently and irreversibly deactivate the smart contract. Tokens must be destroyed/cancelled before deactivation of the contract or during the operation. Once the contract is deactivated, the issuer can no longer create or cancel tokens. This function is necessary to allow the issuer to carry out certain corporate actions (e.g. share splits, reverse splits or mergers), which require that all existing tokens are either cancelled or immobilized and decoupled from the financial instruments (i.e. the tokens no longer represent financial instruments). This functionality can also be used if the issuer decides that it no longer wishes to have its instruments issued in tokenized form. 
10. **Know deactivate status:** For a particular CMTAT token, any person may know whether the contract is deactivated. 

If decimals are permitted:

11. **Know decimals:** For a particular CMTAT token, any person may know the decimals used to get its user representation. 

In the case where decimals are permitted (for example stablecoins or when the law governing the tokenized security allows the transfer of fractions) this function is used to determine how to correctly display the token's amount by indicating where the decimal point is placed when displaying balances. For example, if

decimals equals 2, a balance of 505 tokens should be displayed to a user as 5.05 ($505 / 10^{**2}$).

This functionality is not mandatory if the default decimals state for the token is zero on the target ledger.

Attributes applicable to all CMTAT tokens:

The smart contract must include certain basic information on the tokens, namely:

- Name
- Ticker symbol (optional)
- Token ID (ISIN or other identifier) (optional)
- Reference to any legally required documentation about the distributed ledger or the smart contract, such as the tokenization terms, the terms of the instrument and other relevant documents (e.g. prospectus or key information document).

Note that decimal numbers must be set to zero (which means that the tokens admit no fractional parts), unless the law governing the tokenized security allows the transfer of fractions.

Additional attributes applicable to tokens used for debt securities:

- Guarantor identifier (if applicable)
- Debtholder representative identifier (if applicable)
- Unique identifier / hash (if applicable)
- Issuance date
- Currency of payments (if applicable)
- Par value (principal amount) (if applicable)
- Minimum denomination (if applicable)
- Maturity date (if applicable)
- Interest rate (if applicable)
- Coupon payment frequency (if applicable)
- Interest schedule format (if applicable). The purpose of the interest schedule is to set, in the parameters of the smart contract, the dates on which the interest payments accrue.
 - Format A: start date/end date/period
 - Format B: start date/end date/day of period (e.g. quarter or year)
 - Format C: date 1/date 2/date 3/...
- Interest payment date (if different from the date on which the interest payment accrues):
 - Format A: period (indicating the period between the accrual date for the interest payment and the date on

which the payment is scheduled to be made)

- Format B: specific date
- Day count convention
- Business day convention

3.1.2 Enforcement module mandatory functions

Rationale: The issuer (or a third party appointed by it) must be in a position to freeze a specific ledger address (as opposed to pausing the whole smart contract) in case of suspicious activity.

Functionalities:

12. Freeze: Prevent any token from being transferred to or from a given address until Unfreeze is called. 

13. Unfreeze: Allow transfer of tokens again. 

14. Know frozen status: Allows the issuer and token holder to know whether a particular address is frozen.  

This function must be accessible, at a minimum, to the issuer and the target token holder. On public blockchains, the function should generally be available to everyone, since all data is publicly visible. On private blockchains, such as Aztec, the issuer may want to restrict access to the issuer, the relevant token holder, and possibly any third parties explicitly authorized by the issuer.

§ 3.2 *Optional functionalities*

The following modules and functionalities make it possible to add features that can be used to support certain issuer corporate actions such as dividend distributions or interest payments. The optional modules can also be used for implementing whitelisting rules, transfer restrictions or issuing debt securities.

3.2.1 Snapshot module

Rationale: To carry out certain corporate actions, such as dividend or interest payments, it is necessary to determine the number of tokens held by certain users at a certain point in time. This is particularly important in relation to distributions or the exercise of rights attached to tokenized securities. These moments are generally referred to as the “record date” or “record time” (i.e. the time used to determine the eligibility of security holders for a corporate action). A snapshot determines the number of tokens recorded on various ledger addresses at a specific point in time, and this information can be used to perform on-chain transactions.

When tokenized securities are actively traded, ownership of a security may differ between the time a corporate action is executed (e.g. when dividends or interest are paid) and the time the security holder’s rights arise (e.g. when dividends or interest become due). The Snapshot module enables transactions (on-chain or off-chain) based on the ledger’s state at the record time, not when the corporate action is executed. This function allows a particular action to be executed at a time determined in advance, guaranteeing that each token benefits once (but only once) from the corporate action.

Functionalities:

15. **Schedule a snapshot:** For a particular CMTAT token, the issuer may schedule the creation of a snapshot at a certain time. The time of the newly scheduled snapshot cannot be before the time of the latest scheduled, but not yet created, snapshot. 🏛️
16. **Reschedule a snapshot:** The issuer can change the time of a scheduled snapshot. The new scheduled time cannot be before the time of the previously scheduled snapshot or after the time of the next scheduled snapshot (i.e. scheduled snapshots cannot be reordered). 🏛️
17. **Unschedule a snapshot:** For a particular scheduled snapshot, the issuer can cancel a previously scheduled snapshot. The unscheduled snapshot must be the last scheduled snapshot, and its time must be in the future. 🏛️
18. **Snapshot time:** For a particular scheduled, but not yet created, snapshot, anyone may know the snapshot time. 👤
19. **Snapshot total supply:** For a particular created snapshot, anyone may know the total number of tokens that were in circulation at the snapshot creation time. 👤
20. **Snapshot balance:** For a particular created snapshot and a particular ledger address, anyone may know the number of tokens recorded on the relevant ledger address at the snapshot creation time. 👤

3.2.2 Validation module

Rationale: To limit the scope of persons or entities who may hold the relevant instruments, issuers may decide to implement conditional restrictions on the transfer of the tokenized instruments and to approve or deny transfers based on specific rules. The validation module can be used to implement a specific set of rules that can be embedded into the smart contract or checked against lists. The Validation module ensures that all transactions meet the conditions set in the rules before a transfer can take place. These restrictions are typically implemented when the issuer chooses to apply transfer restrictions or to implement whitelisting rules to ensure compliance with applicable laws, such as anti-money laundering laws or international sanctions.

Functionalities:

21. **Conditional transfer request:** Send a request for validation of a particular transfer to the issuer. 👤
22. **Conditional transfer approve:** Approve / reject a request for validation of a particular transfer. 🏛️
23. **Assign to whitelist:** The issuer identifies addresses to be included in a whitelist. 🏛️

3.2.3 Authorization module

Rationale: Issuers may wish to implement a role-based access control to the token functionalities, rather than distinguish only between issuer and user. This may help reflect the issuer organization's governance model. The Authorization module thus allows the issuer to assign responsibilities and authorizations to various persons (accounts).

Functionalities:

- 24. **Grant role:** Grant a role to a given account. 
- 25. **Revoke role:** Revoke a role from the given account. 
- 26. **Role attribution:** Know whether a given account has a given role. 

3.2.4 Distribution module

Issuers may be required to make distributions to holders of securities (e.g. dividend payments for equity securities or interest payments for debt securities). Issuers may wish to carry out distributions off-chain (i.e. by transferring fiat currencies to the securities' holders' bank account). However, if the issuer intends to carry out such distributions on-chain, this may require the distribution of new tokens to existing token holders, on the basis of a snapshot carried out at the moment the legal entitlement to the distribution arises as not all token holders may be eligible for distributions.

Distribution events are typically performed according to a predefined schedule, and according to the token distribution at a given time, which can be determined by a snapshot performed with the Snapshot module.

Functionalities

- 27. **Distribution create parameters:** Define settlement token (i.e. the token that is to be distributed), identify a (past or future) block time/height for distribution snapshot, and amount to be distributed. 
- 28. **Distribution set eligibility:** Flag a given users' tokens as being eligible or non-eligible to receive distributions (default: eligible). 
- 29. **Distribution set deposit:** Send deposit amount for claiming settlement tokens to token holders flagged as eligible. 
- 30. **Distribution claim deposit:** Allow token holders to claim their share of a deposit, identified by a deposit Identification, according to the token balance at the snapshot created at the defined time/height. 

Additional use cases for tokens representing debt instruments:

- 31. **Distribution schedule:** Define a schedule for interest payments [and repayment of the par value at maturity], based on the token attributes. 
- 32. **Distribution unschedule:** Cancel the previously set schedule for interest payments [and repayment of the principal amount at maturity]. 

3.2.5 Debt module

This module can be used for securities that incorporate rights or claims that are contractual in nature (as opposed to equity interests).

A number of events may occur during the lifetime of a debt instrument. Contrary to corporate actions, some of these events typically involve third parties (e.g. a bond agent or a rating agency).

The issuer may in such cases wish to delegate the relevant functions to the relevant third parties.

Functionalities:

- 33. **Flag as default:** The tokens are flagged as representing debt instruments in respect of which the issuer has

defaulted (bond agent function).

- 34. **Remove default flag:** The tokens are no longer flagged as representing debt instruments in respect of which the issuer has defaulted (bond agent function).
- 35. **Flag as redeemed:** The tokens are flagged as representing debt instruments that have been redeemed (bond agent function).
- 36. **Set rating:** The tokens are flagged as representing debt instruments that have been given a particular rating (bond agent).

3.2.6 Enforcement module optional functionalities

- 37. **Enforce a transfer:** Transfer an amount of tokens without requiring the consent of the holder.
This function can be implemented to comply with a transfer order, for example from a judicial authority.
- 38. **Partial freeze:** Partially freeze the balance of a token holder.
A possible use case of this optional function is intended to avoid a 'double spend' between the execution of a transaction on a trading platform and the settlement of the trade in a distributed ledger. Only the number of tokens actually sold is blocked.
- 39. **Know active balance:** Return the unfrozen balance of an account. This is an optional function linked to the partial freeze function.
- 40. **Know frozen balance:** Return the frozen balance of an account. This is an optional function linked to the partial freeze function.
- 41. **User-approved cancel:** This cancellation function requires authorization from the token holder. It can be implemented if the issuer wishes to distinguish between cancellations to manage supply and those made to carry out a court order. This functionality also allows token holders to cancel their own tokens. 
- 42. **Approve:** This function grants another account (the delegate) permission to transfer a specific amount of tokens from the token account.

Note: While the 'approve' function is not mandatory, it is strongly recommended, as CMTA has been advised that secondary market capability might strongly depend on a delegated approval function in order to automate trading and settlement for any regulated entity. Issuers should consult with the relevant trading and settlement venues if a listing is contemplated at any point in time. This function may be included elsewhere than the enforcement module within an implementation.

4. REFERENCE IMPLEMENTATIONS

The above framework describes the functional capabilities that a token should have in order to represent a financial instrument. Designed to be generic, technology-agnostic, and modular, the framework provides institutions with flexibility in technology choices and the functional requirements. To assist institutions in the practical application of the framework, CMTA has published reference implementations, demonstrating an example of how the functionalities can be implemented. Additional implementations beyond those listed below, which cover multiple design and technology choices, are encouraged.

§ 4.1 *Ethereum*

A reference implementation of the CMTAT is in the Solidity language (suitable for Ethereum, Polygon, and other EVM-based chains) and is available at <https://github.com/CMTA/CMTAT>. This implementation, which generates fungible tokens based on Ethereum's ERC-20 standard, can be used with Ethereum-based wallets, DeFi protocols, and layer-2 protocols, and generally on any blockchain that supports the Solidity coding language.

The Solidity reference implementation supports the CMTAT's mandatory and optional modules outlined above, including:

- BaseModule which contains the smart contract version;
- the "Validation" module, which calls an IERC-1404 "rule engine" for compliance management and the Enforcement and Pause modules;
- Enforcement Module (freeze addresses);
- Pause Module to pause and deactivate the smart contract;
- ERC20BaseModule which contains the main ERC20 related functions;
- the "Authorization" module;
- the SnapshotEngine implementing the "Snapshot" function;

In addition, it includes:

- a module for "gasless" transactions; and
- a "Document" module, which calls an IERC-1643 "engine" for document management.
- the Debt Module (also an extension/option)
- the SnapShotEngineModule to call a SnapShotEngine to perform on-chain snapshots.

The above list is not exhaustive and the complete list can be consulted at <https://github.com/CMTA/CMTAT/tree/dev?tab=readme-ov-file#list>

CMTAT's Solidity reference implementation also includes platform-specific features regarding the upgradeability of the smart contract, which can be achieved through the use of proxy contracts. This feature makes it possible for issuers to update an existing version of the smart contract without having to "burn" the outstanding tokens and "mint" new ones.

In addition, this implementation calls a number of "Engines". Engines are external smart contracts that can be activated (or "called") by specific modules of the CMTAT. The use of external smart contracts makes it possible to add functionalities to tokens without increasing the size of the smart contract that generates them. The CMTA provides open-source versions of four different engines: a Document Engine, a Rule Engine, a SnapShot Engine, and a Debt Engine.

Further information can be found here: <https://github.com/CMTA>

§ 4.2 *Tezos*

An implementation of the CMTAT for Tezos in SmartPy (Python) language is available. The code was developed by the Tezos Foundation and is available here: <https://github.com/CMTA/CMTAT-Tezos-FA2>. CMTA's Technical Committee

reviewed this implementation for compliance with CMTA's standard only.

An implementation of the CMTAT for Tezos is also available in the Ligo coding language (available here: <https://github.com/ligolang/CMTAT-Ligo>). CMTA has not been involved in the development of this implementation.

§ 4.3 Aztec

A privacy-preserving implementation of CMTAT in Noir (Aztec network DSL) is available. This version builds on Aztec's technology on the Ethereum layer-2 protocol Aztec (<https://aztec.network>), and enables confidential transactions using zero-knowledge proofs. The code was developed by Taurus SA: <https://github.com/CMTA/private-CMTAT-aztec>.

§ 4.4 Solana

Specifications and guidelines for deploying a CMTAT token using the Solana Token and Extensions Programs: <https://github.com/CMTA/CMTAT-Solana>

5. CONCLUSION

The CMTAT framework defines the functions required to (i) create tokens that are suitable for association with financial instruments for tokenization purposes, and (ii) manage certain events affecting the relevant financial instruments using on-chain or off-chain mechanisms, as appropriate and divides these functionalities into mandatory and optional functionalities.

The CMTAT is under continuous development, with regular releases to include additional features, improvements in security and updates reflecting best practice and other industry standards.

As regulatory frameworks evolve, the CMTA welcomes feedback on changes to the mandatory or optional functionalities detailed in this framework.

In particular different implementations of the CMTAT may evolve separately from the CMTAT standard framework, adding optional functionalities not necessarily outlined in the framework.

Capital Markets and Technology Association

cmta.ch

Route de Chêne 30
1208 Geneva

admin@cmta.ch