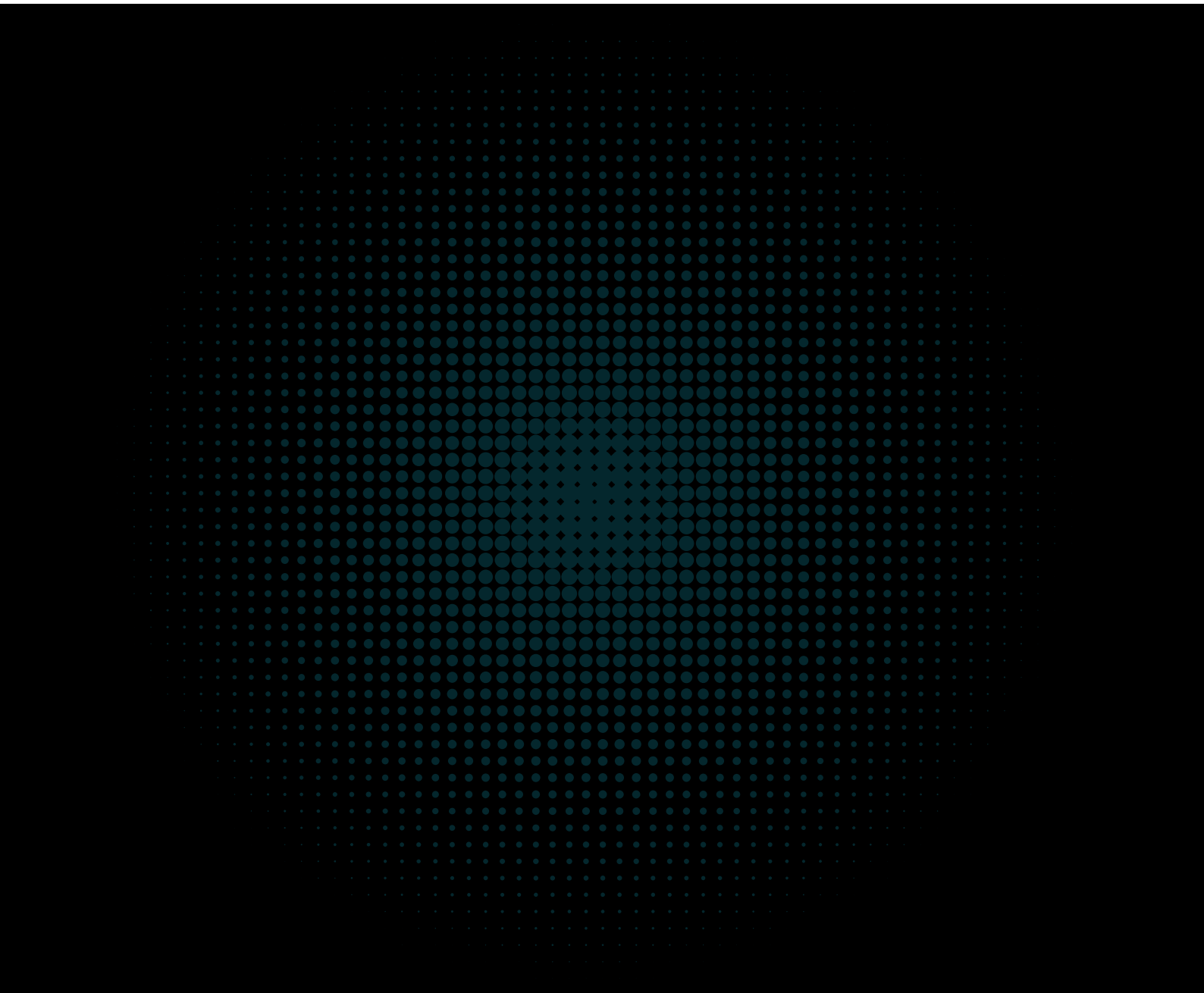
cmta.

CMTAT Specification

Functional specifications of the CMTA Token for the tokenization of financial instruments on Ethereum and EVM compatible blockchains.

CMTAT version: v3.1.0
First published: December 2025



CMTA Token

> To use the CMTAT, we recommend the latest audited version, from the Releases page. Currently, it is the version v3.0.0.
>
> A pdf file of the v3.0.0 README is available here: CMTATSpecificationV3.0.0.pdf

Introduction

The CMTA token (CMTAT) is a security token framework that includes various compliance features such as conditional transfer, account freeze, and token pause, as well as several technical features such as ERC-7802 for cross-chain transfer and ERC-7201 for upgradeadibility.

This repository provides CMTA's reference Solidity implementation of CMTAT, suitable for EVM chains such as Ethereum.

History

The CMTA token (CMTAT) is a security token framework that includes various compliance features such as conditional transfer, account freeze, and token pause. CMTAT was initially optimized for the Swiss law framework, however, these numerous features and extensions make it suitable for other jurisdictions too.

The CMTAT is an open standard from the Capital Markets and Technology Association (CMTA), which gathers organizations from the financial, legal and technology sectors.
The CMTAT was first developed by a working group of CMTA's members and its development is now overseen by the Technical Committee of CMTA's Advisory Board.

Goal

CMTAT has been built with five main goals:

1. Suitable for various regulatory financial assets and instruments (Equities, Structured products, Debt and Stablecoin) and adapted to any jurisdiction (international)
2. Easy to modify and adapt for specific use-case (customization) through its modular architecture
3. Interoperability with the Ethereum ecosystem by implementing recognized standards:
    - Tokenization: ERC-20, ERC-3643 (without on-chain identity), ERC-1404, ERC-7551, ERC-1363,...
    - Technicals: ERC-2771 (MetaTx/Gasless), ERC-7201, ERC-7802...
4. Security by undergoing audits from trusted firms like ADBK and Halborn, and by implementing a range of industry best practices.
    - Strong code statements coverage(~99.43%) with 3078 automated tests executed
    - Run static analyzer (Aderyn, Slither), as well as AI Auditing tool (Nethermind Audit Agent), before and after the audits
    - RBAC Access Control to clearly separates the different roles and permissions
5. Freedom of use through an open-source weak copyleft license (MPL-2.0)

By taking these five main goals, here a comparison with others known implementations to deploy financial instruments on-chain.

	CMTAT	**ERC-3643 (Tokeny implementation)**	**ERC-1400 (UniversalToken)**	**TokenF**	**ERC-20 Smart Coin (Cast framework)**
Version/Commit	v3.0.0 (09/2025)	4.2.0-beta2 (01/2025)	54320c6 (01/2024)	0c1c2ac (04/2025)	dd8bf5e (01/2025)
Company / Association behind	CMTA	Tokeny, ERC3643 Association	Consensys	Distributed Lab	SOCIÉTÉ GÉNÉRALE FORGE
1 (suitable for various financial instruments)	☑	Partial	☑	Partial	Partial

	CMTAT	ERC-3643 (Tokeny implementation)	ERC-1400 (UniversalToken)	TokenF	ERC-20 Smart Coin (Cast framework)
Details	-	Lack of support for Debt product On-chain identity management can potentially make it too complex for stablecoins Also lacks support for adding information related to on-chain terms (hash, uri)	-	Lacks support for adding information related to on-chain terms (hash, uri) as well as Debt product but contracts could be extended.	Lacks support for adding information related to on-chain terms (hash, uri) as well as Debt product
2 (customizable)	☑	☒	☒	☑	☑
Details	Modular architecture	Code difficult to modify because functionalities are not clearly separated and onchain identity management is required	Code difficult to modify because functionalities are not clearly separated	Customizable but uses the Diamant proxy pattern structure which makes it more complex to implement	Contracts are minimalist and easy to modify
3 (interoperability)	☑	☑	Partial	☑	☑
Details	Tokenization: ERC-3643 (without on-chain identity), ERC-1404, ERC-7551, ERC-1363,... Technicals: ERC-20, ERC-2771, ERC-7201, ERC-7802, ...	ERC-20 and ERC-3643	While ERC-1400 is an ERC-20, the standard ERC-1400 is not itself an official standard It has also a dependence with ERC-1820 registry contract, which is not always available/deployed on some layer2.	ERC-20 and ERC-2535	ERC-20
4 (Security)	☑	☑	☒	☒	☒
	- 1.0 and 2.3.0: audited by ABDK 3.0.0 audited by Halborn - RBAC Access Control	- Past version audited by Hacken. - Lack of granularity in term of Access Control (only two roles: Agent and Owner)	No official public audit for the last commits, last audit was done in 2020	No official audit available	No official audit available
5 (License - Open source & Allow Commercial use)	☑	Partial (only ERC-3643 core)	☑	☑	☑

	CMTAT	ERC-3643 (Tokeny implementation)	ERC-1400 (UniversalToken)	TokenF	ERC-20 Smart Coin (Cast framework)
	MPL-2.0 (weak copyleft)	- ERC-3643 core: GPL 3.0 (strong copyleft) - Compliance module: CC-BY-NC-4.0(forbid commercial use)	Apache 2.0 (permissive)	MIT (permissive)	Apache-2.0 license (permissive)

Who uses CMTAT and for what?

CMTAT is suitable for the digitalization of various financial assets. Below is a selection of public case studies

More details are available here: cmta.ch/faqs

Digitalization of equity securities

The CMTAT was initially designed for the digitalization of company shares. For SMEs, digitalization provides an opportunity to access new financing and investment models by selling digital shares through online exchanges. Some companies that have digitalized shares using the CMTAT include:

- Daura uses the CMTAT through their own fork to digitalize the shares of companies using its platform, deployed on zkSync.
- Taurus SA (partial list): Magic Tomato SA (2022) - an online grocery platform opened its governance and capital to its community, by issuing digital non-voting shares (bons de participation), Qoqa Brew (2022) - an online retailer opened the capital of its on-site brewery Q-Brew to its community by issuing digital non-voting shares, Cité Gestion SA (2023) - a Swiss bank and wealth manager, issued digitalized shares in 2022, using the CMTAT, CODE41 (2023) - a Swiss watchmaking company tokenized its shares for a capital increase using CMTAT.

Digitalization of debt securities

- Project Guardian - UBS (2024): CMTAT was used to issue a digital bond by UBS, as part of the first live repo transaction with a natively-issued digital bond on a public blockchain as part of the Monetary Authority of Singapore's (MAS) Project Guardian.
- The Obligate platform Enote Protocol enables BulletBond issuances using smart contracts, deployed on Polygon PoS. For this purpose, they use a fork of CMTAT with the `SnapshotModule` (replaced in CMTAT v3.0.0 by the SnapshotEngine) and the DebtModule.
- SCCF (2023): trade finance firm SCCF issued short term tokenized notes to refinance a loan to a commodity trading firm active in biofuels through Taurus SA. See also SCCF and Taurus Announce Successful Tokenization of End-to-End Trade Finance Transaction on TDX Marketplace (2024)

Digitalization of structured products

- In early 2024, UBS executed a pilot transaction with OSL, a licensed professional investor in Hong Kong, to issue a tokenized warrant on Ethereum using the CMTAT smart contract framework. The tokenization arrangement for this warrant utilizes the CMTAT codebase to represent the warrant smart contract, which forms part of the tokenized register of holders. See ubs.com - UBS expands its digital asset capabilities by launching Hong Kong's first-ever tokenized warrant on the Ethereum network [ubs.com].
- Credit Suisse, Pictet and Vontobel (2022) issued tokenized investment products that were traded on BX Swiss as part of a proof-of-concept leveraging the CMTAT.

Digitalization of artwork

- Syz Group, a Swiss private bank, has successfully digitized two pieces of art using CMTAT in 2023 and 2024. See Syz Art Tokenisation

Tokenized market funds

- In 2024, UBS launched UBS USD Money Market Investment Fund Token (uMINT), a Money Market investment built on Ethereum distributed ledger technology. The tokenization arrangement for this fund utilizes CMTAT codebase to represent the fund smart contract, which forms part of the fund's tokenized register of members. See ubs.com - UBS Asset Management launches its first tokenized investment fund [ubs.com]

Tokenization platform

- Fireblocks integrates CMTAT into their tokenization platform. See also Fireblocks - Fireblocks joins CMTA to define the standards for tokenization in traditional capital markets
- Taurus SA integrates CMTAT (Solidity and Tezos version) into their tokenization platform called Taurus-CAPITAL

Wrapped Tokens

- 21.co (the parent company of 21Shares acquired by FalconX) used CMTAT through their own fork to create Wrapped Tokens on Ethereum. See for example Wrapped Bitcoin(21BTC) on Etherscan and their announcement

Where CMTAT is mentioned?

CMTAT is mentioned in several different reports. While these reports do not take into account the latest changes with the version v.3.0.0, it gives already a good indication of how CMTAT can be used. The points raised by these also allowed for numerous improvements to be made to the CMTAT.

- Forum - Asset Tokenization in Financial Markets: The Next Generation of Value Exchange (2025), page 38
- King's Business School/Rhys Bidder - What Is The Future Of Stablecoins, And How Do We Get There? (2025), page 33
- Nethermind - Tokenization Standards: The Missing Link for Institutional Adoption (2025): page 2, 16, 19, 33-36 & 39
- Project Guardian - Fixed Income Framework (2024): page 13, 39, 59 & 65
- ICMA contribution to MAS Guardian Fixed Income Framework (GFIF) publication (2024)

Use case

Financial instruments

This reference implementation allows the issuance and management of tokens representing equity securities, and other forms of financial instruments such as debt securities and structured products. It can also be used for stablecoins.

Jurisdiction

CMTAT was initially optimized for the Swiss law framework, it then took a more **international** path with the version v3.0.0. Subsequently, its numerous compliance features, as well as the numerous configuration possibilities during deployment, make it also suitable for other jurisdictions.

Its modular structure allows it to be easily modified to suit specific cases. For example, a deployment version has been made for Germany (ERC-7551 / eWpG).

You may modify the token code by adding, removing, or modifying features. However, the core modules must remain in place for compliance with the CMTA specification.

Specific deployment version tailored to use case

Product use case (equities, stablecoins)

CMTAT comes with several different deployment versions to meet specific use cases.

Product	Deployment version	Note
Equities	CMTAT Standard	All features, without those directly to Debt
Equities in Germany	CMTAT ERC-7551	The standard version with a few supplementary functions to meet the standard ERC-7551, tailored for the Germany and eWpg.
Debt/bond	CMTAT Debt	CMTAT Standard is also suitable but this version adds the possibility to put several on-chain information related to debt and bond product
Stablecoin (e.g USDC/USDT)	CMTAT Light	The core features (i.e., minting, burning,address freeze / blacklisting, pause) without additional functions required by equities and debt instruments (e.g., document management, snapshot, partial freeze of balances).

Technical use case (whitelist, upgradeable/proxy)

Features	Deployment version supported
Restrict transfer to inside a whitelist / Allowlist	CMTAT Allowlist Or all other deployment (except Light) version with a `RuleEngine` configured

Features	Deployment version supported
On-chain snapshot (useful for on-chain dividend distribution)	All deployment version (except Light) with a `SnapshotEngine` configured
Deployment through proxy (Upgradeable) Deployment immutable (standalone / without proxy)	Each deployment version comes with a standalone (immutable) or upgradeable mode. A specific deployment version exists for UUPS Proxy
MetaTx/Gasless with ERC-2771	All deployment version, except Debt & Light version

CMTAT for stablecoins

Here is a comparison between the features present in major custodian stablecoin and the library CMTAT.

		Monerium	USDC	USDT	CMTAT 3.0.0 Light	CMTAT 3.0.0 Standard
Source		ec59a36	Ethereum USDC implementation contract	Ethereum USDT address		
Currency		$, euros, pound sterling, Icelandic króna	$	$	-	-
Company behind		Monerium	Circle	Tether	CMTA	CMTA
Standard						
	ERC-20	☑	☑	☑	Same as standard version	☑
	ERC-2612 Permit	☑ (GitHub)	☑	☒	Same as standard version	☒
	ERC-3009 (Transfer With Authorization)	☒	☑	☒	Same as standard version	☒
	ERC-2771 (MetaTX)	☒	☒	☒	☒	☑ (ERC2771Module / CMTATBaseERC2771)
ERC-20 extends functionalities						
	Mint/issue	☒ (see mint with allowance)	☒ (see mint with allowance)	☑	Same as standard version	☑
	Mint with dedicated allowance ("mintFrom")	☑ (Github)	☑	☒	Same as standard version	☒
	Batch Mint version	☒	☒	☒	Same as standard version	☑
	burn / redeem	☑ (Github)	☑	☑ (`redeem`/ `destroyBlackFunds`	Same as standard version	☑
	Set name after deployment	☒	☒	☒	Same as standard version	☑ (ERC20BaseModule)

		Monerium	USDC	USDT	CMTAT 3.0.0 Light	CMTAT 3.0.0 Standard
	Set symbol after deployment	☒	☒	☒	Same as standard version	☑ (ERC20BaseModule)
Regulatory						
	Integrated blacklist (inside token smart contract)	☒	☑	☑	Same as standard version	☑
	External blacklist (can be shared with several different tokens)	☑ GitHub	☒	☒	☒	☑ (through a dedicated smart contract RuleEngine)
		Monerium	**USDC**	**USDT**	**CMTAT 3.0.0 Light**	**CMTAT 3.0 Standard**
Access Control						
	Ownership	☑ (Github)	☑	☑	Same as standard version	☒ (use Access Control instead, but ownership could be added)
	RBAC Access control	☑ GitHub	☑ (Minter & Blacklister)	☒	Same as standard version	☑
Upgradeability						
	Upgradable (transparent/Beacon)	☒	☑	☒	Same as standard version	☑
	Upgradeable UUPS	☑ (GitHub)	☒	☒	☒	☑ (through a dedicated deployment version)
	Migrate function integrated	☒	☒	☑ (because USDT was made before the apparition of proxy architecture)	Same as standard version	☒
Standalone (immutable)		☒	☒	☑	Same as standard version	☑ (through a dedicated deployment version)
Pause transfers		Partial Could use the `validator` contract to pause all transfers (GitHub)	☑	☑	Same as standard version	☑ (PauseModule)
Fee on transfer		☒	☒	☑ (currently set at 0)	Same as standard version	☒

CMTAT for tokenized market funds

Here is a comparison between the features present in known tokenized market funds and the library CMTAT.

		Spiko (EUTBL and USTBL)	Franklin Templeton (FOBXX / Benji)	Blackrock (BUIDL)	CMTAT 3.0.0 Standard	CMTAT 3.00 ERC-1363

		Spiko (EUTBL and USTBL)	Franklin Templeton (FOBXX / Benji)	Blackrock (BUIDL)	CMTAT 3.0.0 Standard	CMTAT 3.00 ERC-1363
Reference			Franklin OnChain U.S. Government Money Fund (FOBXX) Avalanche - Franklin Templeton Launches Tokenized Money Market Fund BENJI On The Avalanche Network	Securitize contracts Proxy Implementation	-	-
Source		9ef58f3	Franklin Templeton Digital Assets - contracts Contract proxy Contract implementation		-	-
Company behind		Spiko	Franklin Templeton	Blackrock through Securitize	CMTA	CMTA
Standard						
	ERC-20	☑	☑	☑	☑	Same as standard version
	ERC-1363	☑	☒	☒	☒	☑
	ERC-2612 Permit	☑ (GitHub)	☒	☒	☒ (Could be extended to support it)	Same as standard version
	ERC-2771 (MetaTX)	☑ (GitHub)	☒	☒	☑	Same as standard version
ERC-20 extends functionalities						
	Mint/issue	☑ (GitHub)	☑	☑	☑	Same as standard version
	Mint with dedicated allowance ("mintFrom")	☒	☒	☒	☒	Same as standard version
	Batch Mint version	☒	☒	☒	☑	Same as standard version
	burn / redeem	☑ (Github)	☑	☑ (burn & omnibusBurn)	☑	Same as standard version
Regulatory						
	Whitelist / Allowlist	☑	☑ (through external contract `moduleRegistry`)	☑ (through external contract `ComplianceServiceWhitelisted`)	☑ (through *RuleEngine*)	Same as standard version
	On-chain country investor restriction /banned	☒	☒	☑ (though an on-chain list of investor and the library `ComplianceServiceLibrary`)		
	Integrated blacklist (inside token smart contract)	☒ (Could be implemented, but use a whitelist system currently)	☒	☒	☑ (EnforcementModule)	Same as standard version
	External blacklist (can be shared with several different tokens)	☑ GitHub	☑ (through external contract `moduleRegistry`)	☒	☑ (RuleEngine)	Same as standard version
	Forced Transfer	☒	☑ (called `instantTransfer`)	☑ (called *size*)	☑	Same as standard version
	Restriction on `transferFrom`	☒	☑ (through `disableERC20ThirdPartyTransfer` & `enableERC20ThirdPartyTransfer`)	☒	Partial (transfer revert if spender is frozen)	Same as standard version
		Spiko	**Franklin Templeton (FOBXX / Benji)**	**Blackrock (BUILD)**	**CMTAT 3.0.0 Standard**	**CMTAT 3.00 ERC-1363**
Access Control						
	Ownership	☑ (Github)	☒ (only `ModuleRegistry` is ownable)	☑	☒ (easily adaptable to support this)	Same as standard version
	RBAC Access control	☑ GitHub	☑	☑ (several roles: Exchange, Issuer, transfer agent and master)	☑	Same as standard version
	Access Control Manager	☑ (GitHub)	☒	☒	☒ (Could be extended to support it)	Same as standard version

		Spiko (EUTBL and USTBL)	Franklin Templeton (FOBXX / Benji)	Blackrock (BUIDL)	CMTAT 3.0.0 Standard	CMTAT 3.00 ERC-1363
Upgradeability						
	Upgradable (transparent/Beacon)	☑	☑	☑	☑	Same as standard version
	Upgradeable UUPS	☑ (GitHub)	☑	☒	☒ (Could be extended to support it)	Same as standard version
Pause transfers		☑ (GitHub)	☑ (trough `enableERC20Transfer` and `disableERC20Transfer`)	☑	☑	Same as standard version
Lock tokens		☒	☒	☑	☒	Same as standard version
Specific functions for omnibus wallet		☒	☒	☑	☒ (see specific deployment version)	Same as standard version
Dedicated function to fetch the list of token holders and their balance		☒	☑ (`getAccountsBalances`)	☑ (`checkWalletsForList` & `balanceOfInvestor`)	☒	Same as standard version
Price indicated on-chain		☒	☑ (`lastKnownPrice`)	☒	☒	Same as standard version

Note

- *Fasanara Capital Ltd* has also tokenized a money market fund. Since they have worked with Tokeny and use therefore the ERC-3643 standard, a comparison with this standard is provided in other sections of this document. See also Tokeny - How Tokeny Powers Fasanara's Tokenized Money Market Funds
- Upgradeability: a specific CMTAT deployment version allows to use UUPS proxy

Comparison of CMTAT and other tokenization frameworks

Here is a comparison between the features present in known tokenization framework and the library CMTAT.

Label	CMTAT Solidity code	ERC-1400	ERC-3643	Cast Framework
Version/implementation compared	CMTAT v3.1.0	UniversalToken (Consensys)	Tokeny's T-Rex 3fcf44d (06/02/2025)	Smart Coin (ERC-20 version) dd8bf5e
Company / Association behind	CMTA	Consensys	Tokeny, ERC3643 Association	SOCIÉTÉ GÉNÉRALE FORGE
ERC-20	☑	☑	☑	☑
Regulatory features				
Transfer restriction (blacklist / address freeze)	☑	☑	☑	☑
Transfer restriction on the spender address (*transferFrom*)	☑	☒	☒	☑ (GitHub)
On-chain identity management	☒ (could be added with a `RuleEngine`)	☒	☑	☒

Label	CMTAT Solidity code	ERC-1400	ERC-3643	Cast Framework
Document management	☑ (ERC-1643)	☑ (ERC-1643)	☒	☒
Whitelist management	☑ (deployment version or RuleEngine)	☑	☑ (on-chain id)	☒
Token contract pause	☑	☑	☑	☑
Conditional Transfer for specific addresses	☒	☒	☒	☑ (integrated into the token contract)
Conditional Transfer for all addresses	☑ (through RuleEngine)	☒	☑ (through compliance contract)	☒
Technical features				
Configurable ERC-20 decimals	☑	☒ Set at 18 (Github)	☑	☒ (18 by default)
Role-based access control	☑	☑	Partial (only one role Agent)	☑
Adaptable access control (code can be easily adapted to other type of access control)	☑	☒	☒	Partial
Mint & burn to any address	☑	☑	☑	☑
Forced transfer function	☑	☑	☑	Partial Only force burn is available (GitHub)
Partially fungible token support	☒	☑	☒	☒
Contract version on-chain	☑	☑	☑	☑ (GitHub)
Deployable on Layer2 and other EVM blockchains	☑ (require PUSH0)	Partial Requires ERC-1820 Registry contract	☑ (require PUSH0)	☑
Other				
License	MPL 2.0 (weak copyleft)	Apache 2.0 (permissive)	GPL 3.0 (strong copyleft)	Apache 2.0 (permissive)
Third-party security audit	☑ (ABDK & Halborn)	☒	☑ Hacken)	☒
CMTAT unique features				
Regulatory features				
Security identifiers	☑	☒	☒	☒
Explicit support of debt instruments	☑	☒	☒	☒
Technical				
MetaTx ("Gasless") support (ERC-2771)	☑	☒	☒	☒
Customizable modular design	☑	☒	☒	☒

Label	CMTAT Solidity code	ERC-1400	ERC-3643	Cast Framework
ERC-20 custom errors (ERC-6093)	☑ (use OpenZeppelin v5)	☒	☒ (use OpenZeppelin v4)	☒ (use OpenZeppelin v4)
ERC-5679: Token minting and Burning	☑	☒	☒	☒
Upgradibility with ERC-7201	☑	☒	☒	☒
Snapshots/checkpoints	☑ (external contract or by extending CMTAT)	☒	☒	☒
Cross-chain bridge				
ERC-7802 Cross-chain transfer	☑	☒	☒	☒
Chainlink CCT support	☑	☒ (Lack burn/burnFrom/mint)	Partial (Lack burn/burnFrom)	☒ (Lack owner/getCCIPAdmin/burnFrom)

Note

- At the time of our analysis (July 2025), the next version of T-REX/ERC-3643 had not yet been merged into the main branch and officially released. However, we assumed that it would be merged soon and that it would also be audited.
- Access control: CMTAT Access control is easily adaptable because it is implemented in high level contracts (base modules) instead of low level modules (wrapper).

Technical

Security and contribution

The design and security of the CMTAT was supported by ABDK (CMTAT v1.0 and v2.3.0) and Halborn (CMTAT v3.0.0), two leading audit companies in smart contract security.

- The preferred way to receive comments is through the GitHub issue tracker.
- Private comments and questions can be sent to the CMTA secretariat at admin@cmta.ch.
- For security matters, please see SECURITY.md.

Overview

> Core means that they are the main features to build CMTAT

Core features

The CMTAT supports the following core features:

- ERC-20:
  - Mint, burn, and transfer operations
  - Configure `name`, `symbol` and `decimals` at deployment, as well as ERC-3643 functions to update `name` and `symbol` once deployed
- Pause of the contract and mechanism to deactivate it
- Freeze of specific accounts through ERC-3643 functions.

Extended features

> Extended features are nice-to-have features. They are generally included in the majority of deployment version.

The CMTAT supports the following extended features:

- Add information related to several documents (ERC-1643) though an external contract (`DocumentEngine`)
- Perform snapshot on-chain through an external contract (`SnapshotEngine`)
- Conditional transfers, via an external contract (`RuleEngine`)
- Put several information on-chain such as `tokenId` (ISIN or other identifier), `terms` (reference to any legally required documentation) and additional information (`information`)

Optional features

> Optional means that they are generally specific to deployment version

The CMTAT supports the following optional features:

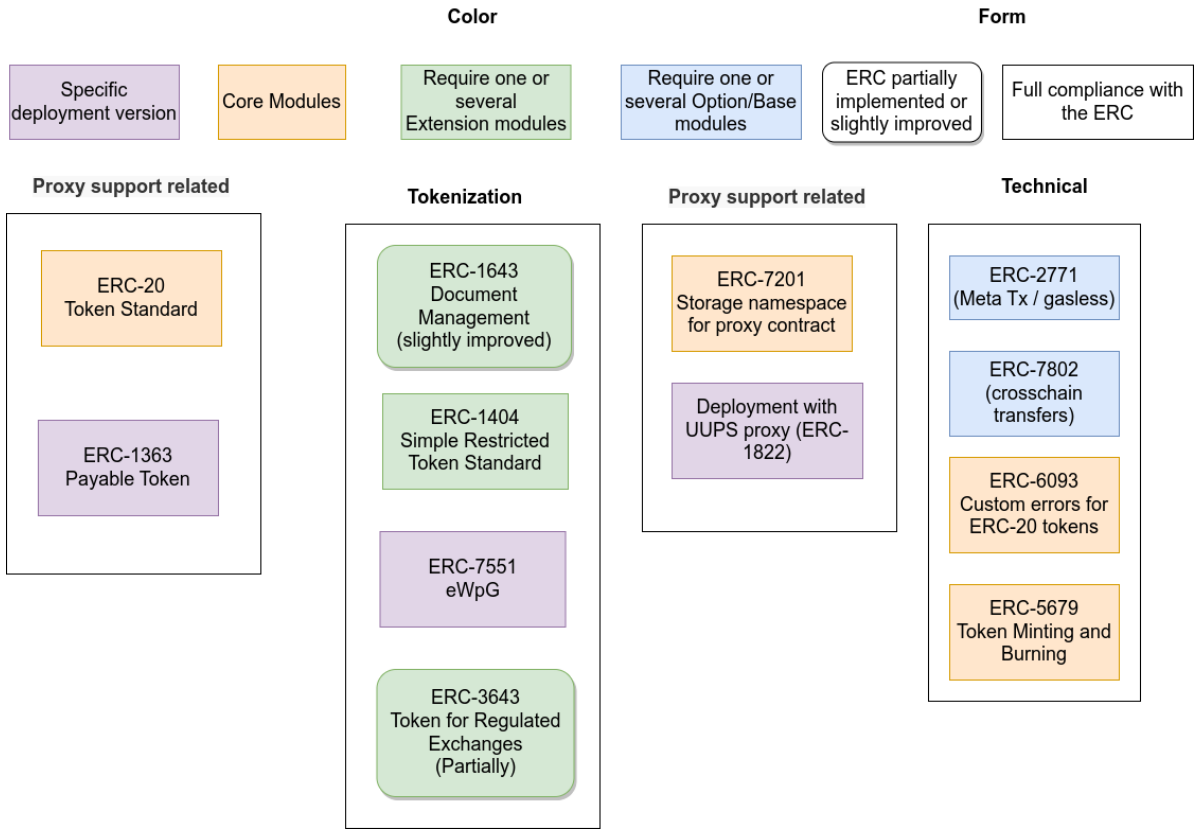
- Transfer restriction through allowlisting/whitelisting (can be also done with a `RuleEngine`)
    - Deployment: CMTAT Standalone Allowlist / CMTAT Upgradeable Allowlist
    - Module: AllowlistModule
- Put Debt information and Credit Events on-chain
    - Deployment: CMTAT Standalone Debt / CMTAT Upgradeable Debt
    - Module: DebtModule & DebtEngineModule
- Cross-chain functionalities with ERC-7802
    - Define directly in a CMTAT Base contract (not a module)
- "Gasless" (MetaTx) transactions with ERC-2771
    - Module: ERC2771Module

Furthermore, the present implementation uses standard mechanisms in order to support `upgradeability`, via deployment of the token with a proxy by implementing ERC-7201

Standard ERC

Here the list of ERC used by CMTAT v3.0.0

Schema



CMTAT version support

Here the list of ERC supported between different version:

	Associated contracts/modules	**ERC status**	**CMTAT v1.0.0**	**CMTAT v2.3.0**	**CMTAT v3.0.0**	**CMTAT v3.1.0**					
Deployment version					(Standalone & Proxy)		Light	UUPS	ERC1363	Allowlist (whitelist)	Debt
Fungible tokens											
ERC-20	ERC20BaseModule	Standard Track (final)	☑	☑	☑	☑	☑	☑	☑	☑	☑
ERC-1363	CMTATBaseERC1363	Standard Track (final)	☒	☒	☒	☒	☒	☒	☑	☒	☒
Tokenization											
ERC-1404 (Simple Restricted Token Standard)	ValidationModuleERC1404 (Exensions)	Draft	☑	☑	☑	☑	☒	☑	☑	☒	☒
ERC-1643 (Document Management Standard) (Standard from ERC-1400) (Slightly improved)	DocumentModule (Exensions)	Draft	☒	☒	☑ (through DocumentEngine with small improvement)	☑ (through DocumentEngine with small improvement)	☒	☑	☑	☑	☑
ERC-3643 (Without on-chain identity)	Core + ERC20EnforcementModule (extensions)	Standard Track (final)	☒	☒	☑	☑	☒	☑	☑	☑	☑
ERC-7551	Core + ERC20EnforcementModule (extensions)	Draft	☒	☒	☑	☑	Partially	☑	☑	☑	☑
Proxy support related											
Deployment with a UUPS proxy (ERC-1822)	-	Stagnant (but used)	☒	☒	☒	☒	☒	☑	☒	☒	☒
ERC-7201 (Storage namespaces for proxy contract)	All	Standard Track (final)	☒	☒	☑	☑	☑	☑	☑	☑	☑
Technical											

	Associated contracts/modules	ERC status	CMTAT v1.0.0	CMTAT v2.3.0	CMTAT v3.0.0	CMTAT v3.1.0					
ERC-2771 (Meta Tx / gasless)	ERC2771Module (options)	Standard Track (final)	☑	☑	☑	☑	☒	☑	☑	☑	☒
ERC-6093 (Custom errors for ERC-20 tokens)	-	Standard Track (final)	☒	☒	☑	☑	☑	☑	☑	☑	☑
ERC-7802 (cross-chain token/transfers)	ERC20CrossChainModule (options)	Draft	☒	☒	☑	☑	☒	☑	☑	☒	☒
ERC-5679 (Token minting and burning with data)	ERC20MintModule ERC20BurnModule	Standard Track (final)	☒	☒	Partial (without ERC-165 support)	☑	☑	☑	☑	☑	☑

Details

ERC-3643

> ERC specification
> Status: Standards Track

The ERC-3643 is an official Ethereum standard, unlike ERC-1400 and ERC-1404. This standard, also built on top of ERC-20, offers a way to manage and perform compliant transfers of security tokens.

ERC-3643 enforces identity management as a core component of the standards by using a decentralized identity system called onchainid.

While CMTAT does not include directly the identity management system, it shares with ERC-3643 many of the same functions. The interface is available in IERC3643Partial.sol

If you want to use CMTAT to create a version implementing all functions from ERC-3643, you can create it through a dedicated deployment version (like what has been done for UUPS and ERC-1363).

The implemented interface is available in IERC3643Partial.

The main reason the argument names change is because CMTAT relies on OpenZeppelin to name the arguments.

All functions

```
// interface IERC3643Pause
// PauseModule
function paused() external view returns (bool)
function pause() external
function unpause() external

// interface IERC3643ERC20Base
// ERC20BaseModule
function setName(string calldata name) external
function setSymbol(string calldata symbol) external

// IERC3643BatchTransfer
// ERC20MintModule
function batchTransfer(address[] calldata tos,uint256[] calldata values)
external returns (bool)

// IERC3643Version
```

```
// VersionModule
function version() external view returns (string memory)

// IERC3643Enforcement
// EnforcementModule
function isFrozen(address account) external view returns (bool)
function setAddressFrozen(address account, bool freeze) external
function batchSetAddressFrozen(address[] calldata accounts, bool[] calldata
freeze) external;

// IERC3643ERC20Enforcement
// ERC20EnforcementModule
function getFrozenTokens(address account) external view returns (uint256);
function freezePartialTokens(address account, uint256 value) external;
function unfreezePartialTokens(address account, uint256 value) external;
function forcedTransfer(address from, address to, uint256 value) external
returns (bool);


// IERC3643Mint
// MintModule
function mint(address account, uint256 value) external;
function batchMint( address[] calldata accounts,uint256[] calldata values)
external;

// IERC3643Burn
// BurnModule
function burn(address account, uint256 value) external;
function batchBurn(address[] calldata accounts,uint256[] calldata values)
external;

// IERC3643ComplianceRead
// ValidationModuleCore
function canTransfer(
        address from,
        address to,
        uint256 value
    ) external view returns (bool isValid);
}
```

Functions not implemented

All functions related to on-chain identity are **not** implemented inside CMTAT:

- `setOnchainID`
- `setIdentityRegistry`
- `recoveryAddress` because this function takes the `investorOnchainID` as an argument

These following functions to reduce contract code size:

- `batchForcedTransfer` to reduce contract code size
- `batchFreezePartialTokens` and `batchUnfreezePartialTokens`

All functions related to the interface `IAgentRole` because CMTAT uses a RBAC Access Control to offer more granularity in terms of access control.

And finally `setCompliance` because CMTAT uses a different architecture for its `ruleEngine`.

Version

Module: VersionModule

ERC-3643	CMTAT v3.0.0	Deployment version
`version() external view returns (string memory version_)`	Same	All

Pause

Module: PauseModule

ERC-3643	CMTAT v3.0.0	Deployment version
`pause() external`	Same	All
`unpause() external`	Same	All
`paused() external view returns (bool);`	Same	All
`event Paused(address _userAddress);`	`event Paused(address account)`	All
`event Unpaused(address _userAddress);`	`event Unpaused(address account)`	All

ERC20Base

ERC-3643	CMTAT v3.0.0	Deployment version
`setName(string calldata _name) external;`	`setName(string calldata name_)`	All
`setSymbol(string calldata _symbol) external`	`setSymbol(string calldata symbol_)`	All

Supply Management (burn/mint)

ERC-3643	CMTAT v3.0.0 Modules	CMTAT v3.0.0 Functions	Deployment version
`batchMint(address[] calldata _toList, uint256[] calldata _amounts) external;`	ERC20MintModule	`mint(address account, uint256 value)`	All

ERC-3643	CMTAT v3.0.0 Modules	CMTAT v3.0.0 Functions	Deployment version
`batchMint(address[] calldata _toList, uint256[] calldata _amounts) external;`	ERC20MintModule	`batchMint(address[] calldata accounts,uint256[] calldata values)`	All
`batchTransfer(address[] calldata _toList, uint256[] calldata _amounts) external;`	ERC20MintModule	`batchTransfer(address[] calldata tos,uint256[] calldata values)`	All
`burn(address _userAddress, uint256 _amount) external`	ERC20BurnModule	`function burn(address account,uint256 value)`	All
`batchBurn(address[] calldata _userAddresses, uint256[] calldata _amounts) external`	ERC20BurnModule	`batchBurn(address[] calldata accounts,uint256[] calldata values)`	All

Warning: `batchTransfer` is restricted to the MINTER_ROLE to avoid the possibility to use non-standard function to move tokens.

ERC20Enforcement

ERC-3643	CMTAT v3.0.0	Deployment version
`isFrozen(address _userAddress)`	`isFrozen(address account)`	All
`forcedTransfer(address _from, address _to, uint256 _amount) external returns (bool)`	`forcedTransfer(address from, address to, uint256 value) external returns (bool)`	All except Light version (replaced by `forcedBurn`)
`batchForcedTransfer(address[] calldata _fromList, address[] calldata _toList, uint256[] calldata _amounts) external`	Not implemented	-

ValidationModuleCore

Note: `canTransfer` is defined for the compliance contract in ERC-3643.

ERC-3643	CMTAT v3.0.0	Deployment version
`canTransfer(address _from, address _to, uint256 _amount) external view returns (bool)`	`canTransfer(address from, address to, uint256 value)`	All

ERC-7551 (eWPG)

> ERC specification / Ethereum magician
>
> Status: draft

ERC-7551 (eWpG) is a proposal by the German Federal Association of Crypto Registrars for a smart contract interface tailored to *crypto securities* in Germany (eWpG). It aims to provide a flexible, minimal foundational layer so that tokens can meet legal/compliance requirements while leaving the door open on how to restrict the use of the token (on-chain id like ERC-3643).

The implemented interface is available in IERC7551.

Only the specific deployment version dedicated (CMTATERC7551) implements the full interface.

Link to off-chain document

Contrary to the ERC-7551 specification, CMTAT does not enforce a non-zero value for the `termsHash`.

`unpause` will not revert if `termsHash == 0x0`.

Summary tab

The following table compares the functionalities and details how the relevant functionalities are implemented in CMTAT's reference Solidity implementation:

N°	Functionalities	ERC-7551 Functions	CMTAT v3.0.0 standard	CMTAT v3.0.0 ERC7551	Implementations details	Modules
1	Freeze and unfreeze a specific amount of tokens	`freezePartialTokens(address account, uint256 amount, bytes calldata data)` `unfreezePartialTokens(address account, uint256 amount, bytes calldata data)`	☑	☑	-	EnforcementModule (core) ERC20EnforcementModule (extensions)
2	Pausing transfers The operator can pause and unpause transfers	`pause()/unpause()`	☑	☑	-	PauseModule (core)
3	Link to off-chain document Add the hash of a document	`setTerms(bytes32 _hash, string calldata _uri)`	☒	☑	The hash is put in the field `Terms` Terms is represented as a Document (name, uri, hash, last on-chain modification date) based on ERC-1643.	ERC7751Module (options)
	Equivalent functionality		☑	☑	`setTerms(IERC1643CMTAT.DocumentInfo calldata terms_)`	ExtraInformationModule (extensions)
4	Metadata JSON file	`setMetaData(string calldata _metadata)`	☒ (can be put in the field `information`)	☑	-	ERC7751Module (options)
5	Forced transfers Transfer `amount` tokens to `to` without requiring the consent of `from`	`forcedTransfer(address account, address to, uint256 value, bytes calldata data)`	☑	☑	-	ERC20EnforcementModule
6	Token supply management Reduce the balance of `tokenHolder` by `amount` without increasing the amount of tokens of any other holder	`burn(address tokenHolder, uint256 amount, bytes calldata data)`	☑	☑	-	BurnModule (core)
7	Token supply management Increase the balance of `to` by `amount` without decreasing the amount of tokens from any other holder.	`mint(address to, uint256 amount, bytes calldata data)`	☑	☑	-	MintModule (core)
View/read-only functions						

N°	Functionalities	ERC-7551 Functions	CMTAT v3.0.0 standard	CMTAT v3.0.0 ERC7551	Implementations details	Modules
8	Transfer compliance Check if a transfer is valid	`canTransfer(address from, address to, uint256 value)` `canTransferFrom()`	☑	☑	-	ValidationModuleCore
9	Transfer compliance Check if a transfer with a spender is valid	`canTransferFrom(address spender, address from, address to, uint256 value)`	☑	☑	-	ValidationModuleCore
10	Active/Frozen Balance	`getActiveBalanceOf(address tokenHolder)` `getFrozenTokens(address tokenHolder)`	☑	☑	-	ERC20EnforcementModule

Fulls functions

```
// IERC7551Mint
// MintModule
event Mint(address indexed minter, address indexed account, uint256 value,
bytes data);
function mint(address account, uint256 value, bytes calldata data) external;

// IERC7551Burn
// BurnModule
event Burn(address indexed burner, address indexed account, uint256 value,
bytes data);
function burn(address account, uint256 amount, bytes calldata data) external;

// IERC7551Pause
// PauseModule
function paused() external view returns (bool);
function pause() external;
function unpause() external;

// IERC7551ERC20Enforcement
// ERC20EnforcementModule
function getActiveBalanceOf(address account) external view returns (uint256);
function getFrozenTokens(address account) external view returns (uint256);
function freezePartialTokens(address account, uint256 amount, bytes memory
data) external;
function unfreezePartialTokens(address account, uint256 amount, bytes memory
data) external;
function forcedTransfer(address account, address to, uint256 value, bytes
calldata data) external returns (bool);


// IERC7551Compliance is IERC3643ComplianceRead
// ValidationModuleCore
function canTransferFrom(
        address spender,
        address from,
        address to,
        uint256 value
    )  external view returns (bool);
// From IERC3643ComplianceRead
function canTransfer(address from, address to, uint256 value) external view
returns (bool);
```

```
// IERC7551Document
// IERC7551Module
event Terms(bytes32 hash_, string uri_);
event MetaData(string newMetaData);
function termsHash() external view returns (bytes32);
function setTerms(bytes32 _hash, string calldata _uri) external;
function metaData() external view returns (string memory);
function setMetaData(string calldata metaData_) external;
```

ERC-7802 (Crosschain transfers)

> [ERC specification]
> Status: draft

This standard introduces a minimal and extendable interface, `IERC7802`, for tokens to enable standardized crosschain communication.

CMTAT implements this standard in the option module `ERC20CrossChain`

This standard is notably used by Optimism to provide cross-chain bridge between Optimism chain, see [docs.optimism.io/interop/superchain-erc20]

More information available in the cross chain section.

Deployment version: since it is an option module, it is not currently used in the deployment version `debt`, `light` & `allowlist`.

```
interface IERC7802 is IERC165 {
    /// @notice Emitted when a crosschain transfer mints tokens.
    event CrosschainMint(address indexed to, uint256 value, address indexed
sender);

    /// @notice Emitted when a crosschain transfer burns tokens.
    event CrosschainBurn(address indexed from, uint256 value, address indexed
sender);

    /// @notice Mint tokens through a crosschain transfer.
    function crosschainMint(address to, uint256 value) external;

    /// @notice Burn tokens through a crosschain transfer.
    function crosschainBurn(address from, uint256 value) external;
}
```

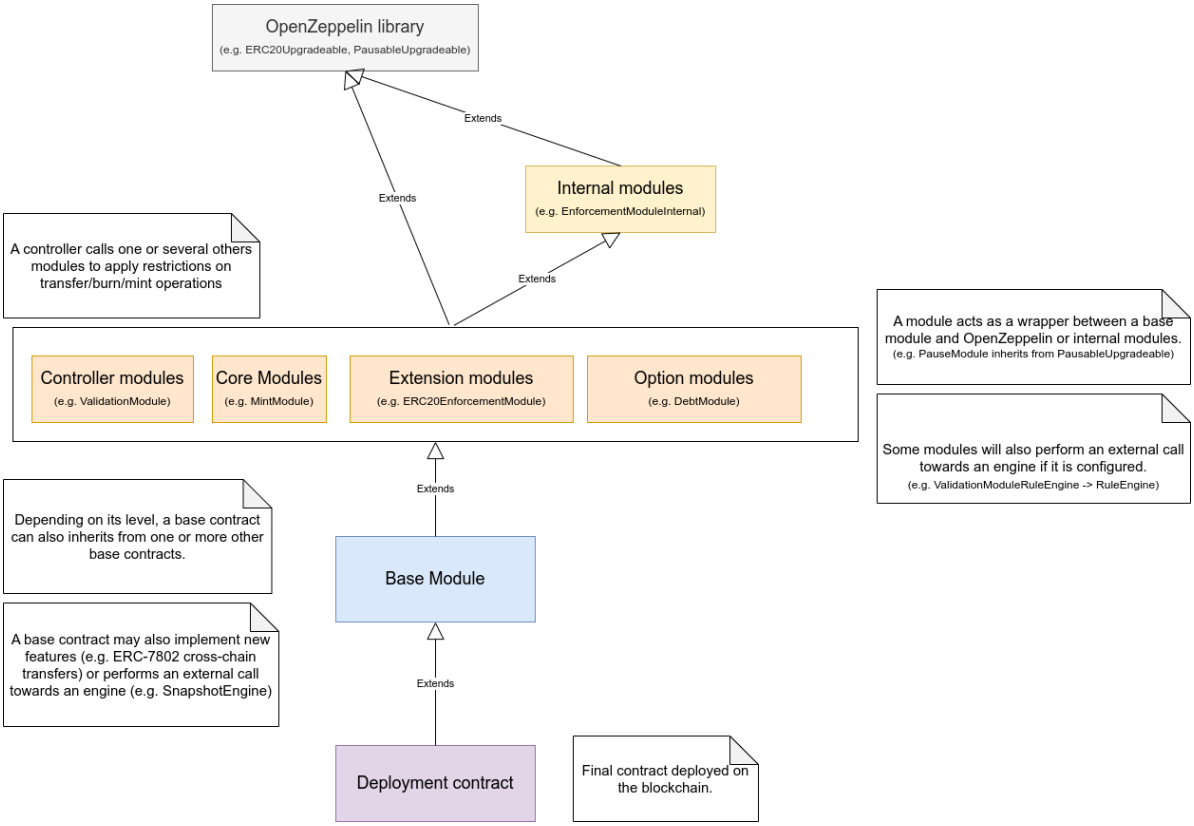
Architecture

CMTAT architecture is divided in two main components: modules and engines.

Overview

Schema

Here is an overview on how CMTAT is build:



Tree file structure

Here is the GitHub file structure for CMTAT repository.

- Contracts

```
├── deployment
│   ├── allowlist
│   │   ├── CMTATStandaloneAllowlist.sol
│   │   └── CMTATUpgradeableAllowlist.sol
│   ├── CMTATStandalone.sol
│   ├── CMTATUpgradeable.sol
│   ├── CMTATUpgradeableUUPS.sol
│   ├── debt
│   │   ├── CMTATStandaloneDebt.sol
│   │   └── CMTATUpgradeableDebt.sol
│   ├── ERC1363
│   │   ├── CMTATStandaloneERC1363.sol
│   │   └── CMTATUpgradeableERC1363.sol
│   ├── ERC7551
│   │   ├── CMTATStandaloneERC7551.sol
│   │   └── CMTATUpgradeableERC7551.sol
│   └── light
│       ├── CMTATStandaloneLight.sol
│       └── CMTATUpgradeableLight.sol
├── interfaces
│   ├── engine
```

```
│   │   ├── IDebtEngine.sol
│   │   ├── IDocumentEngine.sol
│   │   ├── IRuleEngine.sol
│   │   └── ISnapshotEngine.sol
│   ├── modules
│   │   ├── IAllowlistModule.sol
│   │   ├── IDebtModule.sol
│   │   ├── IDocumentEngineModule.sol
│   │   └── ISnapshotEngineModule.sol
│   ├── technical
│   │   ├── ICMTATConstructor.sol
│   │   ├── IERC20Allowance.sol
│   │   ├── IERC5679.sol
│   │   ├── IERC7802.sol
│   │   ├── IGetCCIPAdmin.sol
│   │   └── IMintBurnToken.sol
│   └── tokenization
│       ├── draft-IERC1404.sol
│       ├── draft-IERC1643CMTAT.sol
│       ├── draft-IERC1643.sol
│       ├── draft-IERC7551.sol
│       ├── ICMTAT.sol
│       └── IERC3643Partial.sol
├── libraries
│   └── Errors.sol
├── mocks
│   ├── DebtEngineMock.sol
│   ├── DocumentEngineMock.sol
│   ├── ERC1363ReceiverMock.sol
│   ├── ERC721MockUpgradeable.sol
│   ├── library
│   │   └── snapshot
│   │       ├── ICMTATSnapshot.sol
│   │       ├── SnapshotErrors.sol
│   │       └── SnapshotModuleBase.sol
│   ├── MinimalForwarderMock.sol
│   ├── readme.txt
│   ├── RuleEngine
│   │   ├── CodeList.sol
│   │   ├── interfaces
│   │   │   ├── IRuleEngineMock.sol
│   │   │   └── IRule.sol
│   │   ├── RuleEngineMock.sol
│   │   ├── RuleMockMint.sol
│   │   └── RuleMock.sol
│   ├── SnapshotEngineMock.sol
│   └── test
│       └── proxy
│           ├── CMTAT_PROXY_TEST.sol
│           └── CMTAT_PROXY_TEST_UUPS.sol
└── modules
    ├── 0_CMTATBaseCommon.sol
    ├── 0_CMTATBaseCore.sol
    ├── 0_CMTATBaseGeneric.sol
    ├── 1_CMTATBaseAllowlist.sol
```

```
    ├── 1_CMTATBaseRuleEngine.sol
    ├── 2_CMTATBaseDebt.sol
    ├── 2_CMTATBaseERC1404.sol
    ├── 3_CMTATBaseERC20CrossChain.sol
    ├── 4_CMTATBaseERC2771.sol
    ├── 5_CMTATBaseERC1363.sol
    ├── 5_CMTATBaseERC7551.sol
    ├── internal
    │   ├── AllowlistModuleInternal.sol
    │   ├── common
    │   │   └── EnforcementModuleLibrary.sol
    │   ├── EnforcementModuleInternal.sol
    │   ├── ERC20BurnModuleInternal.sol
    │   ├── ERC20EnforcementModuleInternal.sol
    │   ├── ERC20MintModuleInternal.sol
    │   └── ValidationModuleRuleEngineInternal.sol
    └── wrapper
        ├── controllers
        │   ├── ValidationModuleAllowlist.sol
        │   └── ValidationModule.sol
        ├── core
        │   ├── EnforcementModule.sol
        │   ├── ERC20BaseModule.sol
        │   ├── ERC20BurnModule.sol
        │   ├── ERC20MintModule.sol
        │   ├── PauseModule.sol
        │   ├── ValidationModuleCore.sol
        │   └── VersionModule.sol
        ├── extensions
        │   ├── DocumentEngineModule.sol
        │   ├── ERC20EnforcementModule.sol
        │   ├── ExtraInformationModule.sol
        │   ├── SnapshotEngineModule.sol
        │   └── ValidationModule
        │       ├── ValidationModuleERC1404.sol
        │       └── ValidationModuleRuleEngine.sol
        ├── options
        │   ├── AllowlistModule.sol
        │   ├── CCIPModule.sol
        │   ├── DebtEngineModule.sol
        │   ├── DebtModule.sol
        │   ├── ERC20CrossChain.sol
        │   ├── ERC2771Module.sol
        │   └── ERC7551Module.sol
        └── security
            └── AccessControlModule.sol

29 directories, 93 files
```

- Docs

```
.
├── audits
│   ├── ABDK_CMTA_CMTATRuleEngine_v_1_0
│   │   ├── ABDK_CMTA_CMTATRuleEngine_v_1_0.pdf
```

```
│   │   ├── Taurus.Audit3.1.CollectedIssues.ods
│   │   └── Taurus. Audit 3.3. Collected.ods
│   ├── ABDK-CMTAT-audit-20210910
│   │   ├── ABDK-CMTAT-audit-20210910.pdf
│   │   ├── CMTAT-Audit-20210910-summary.odt
│   │   └── CMTAT-Audit-20210910-summary.pdf
│   └── tools
│       ├── aderyn
│       │   └── v3.0.0-aderyn-report.md
│       ├── mythril
│       │   └── v2.5.0
│       │       ├── myth_proxy_report.md
│       │       └── myth_standalone_report.md
│       └── slither
│           ├── v2.3.0-slither-report.md
│           ├── v2.5.0-slither-report.md
│           └── v3.0.0-slither-report.md
├── general
│   ├── contract-size.png
│   ├── coverage.png
│   ├── crosschain-bridge-support.md
│   └── FAQ.md
├── modules
│   ├── base
│   │   ├── 0_CMTATBaseCore.md
│   │   ├── 3_CMTATERC20CrossChain.md
│   │   └── surya_report
│   ├── controllers
│   │   ├── surya_report_ValidationModuleAllowlist.sol.md
│   │   ├── surya_report_ValidationModuleCore.sol.md
│   │   ├── surya_report_ValidationModuleERC1404.sol.md
│   │   ├── surya_report_ValidationModuleRuleEngineInternal.sol.md
│   │   ├── surya_report_ValidationModuleRuleEngine.sol.md
│   │   ├── surya_report_ValidationModule.sol.md
│   │   ├── validationAllowlist.md
│   │   ├── validation.md
│   │   └── validationRuleEngine.md
│   ├── core
│   │   ├── Base
│   │   │   ├── base.md
│   │   │   └── surya_report_VersionModule.sol.md
│   │   ├── Enforcement
│   │   │   ├── enforcement.md
│   │   │   ├── surya_report_EnforcementModuleInternal.sol.md
│   │   │   ├── surya_report_EnforcementModuleLibrary.sol.md
│   │   │   └── surya_report_EnforcementModule.sol.md
│   │   ├── ERC20Base
│   │   │   ├── ERC20base.md
│   │   │   └── surya_report_ERC20BaseModule.sol.md
│   │   ├── ERC20Burn
│   │   │   ├── ERC20Burn.md
│   │   │   ├── surya_report_ERC20BurnModuleInternal.sol.md
│   │   │   └── surya_report_ERC20BurnModule.sol.md
│   │   ├── ERC20Mint
│   │   │   ├── ERC20Mint.md
```

```
│   │   │   ├── surya_report_ERC20MintModuleInternal.sol.md
│   │   │   └── surya_report_ERC20MintModule.sol.md
│   │   └── Pause
│   │       ├── pause.md
│   │       └── surya_report_PauseModule.sol.md
│   ├── deployment
│   │   └── surya_report
│   ├── extensions
│   │   ├── documentEngine
│   │   │   ├── document.md
│   │   │   └── surya_report_DocumentEngineModule.sol.md
│   │   ├── ERC20Enforcement
│   │   │   ├── erc20enforcement.md
│   │   │   ├── surya_report_ERC20EnforcementModuleInternal.sol.md
│   │   │   └── surya_report_ERC20EnforcementModule.sol.md
│   │   ├── ExtraInformation
│   │   │   ├── extraInformation.md
│   │   │   └── surya_report_ExtraInformationModule.sol.md
│   │   └── snapshotEngine
│   │       ├── Snapshot.md
│   │       └── surya_report_SnapshotEngineModule.sol.md
│   ├── options
│   │   ├── allowlist
│   │   │   ├── allowlist.md
│   │   │   ├── surya_report_AllowlistModuleInternal.sol.md
│   │   │   └── surya_report_AllowlistModule.sol.md
│   │   ├── debt
│   │   │   ├── debt.md
│   │   │   └── surya_report_DebtModule.sol.md
│   │   ├── debtEngine
│   │   │   ├── debtEngine.md
│   │   │   └── surya_report_DebtEngineModule.sol.md
│   │   ├── erc2771
│   │   │   ├── erc2771.md
│   │   │   └── surya_report_ERC2771Module.sol.md
│   │   └── erc7551
│   │       ├── erc7551.md
│   │       └── surya_report_ERC7551Module.sol.md
│   └── security
│       ├── access.md
│       └── surya_report_AccessControlModule.sol.md
├── schema
│   ├── accessControl
│   │   ├── RBAC-diagram.drawio
│   │   └── RBAC-diagram-RBAC.drawio.png
│   ├── drawio
│   │   ├── architecture.drawio
│   │   ├── architecture-ERC.drawio.png
│   │   ├── architecture.pdf
│   │   ├── Engine-AuthorizationEngine.drawio.png
│   │   ├── Engine-DebtVault.drawio.png
│   │   ├── Engine.drawio
│   │   ├── Engine-Engine.drawio.png
│   │   ├── Engine-RuleEngine-Base.drawio.png
│   │   ├── Engine-RuleEngine.drawio.png
```

```
│   │   ├── RuleEngine.drawio
│   │   ├── RuleEngine.png
│   │   ├── transfer_restriction-allowlist.drawio.png
│   │   ├── transfer_restriction.drawio
│   │   └── transfer_restriction.drawio.png
│   └── uml
├── script
│   ├── script_surya_graph.sh
│   ├── script_surya_inheritance.sh
│   └── script_surya_report.sh
├── test
│   ├── coverage
│   └── coverage.json
└── USAGE.md
```

Base contract

The base contracts are abstract contracts, so not directly deployable, which inherits from several different modules.

Base contracts are used by the different deployable contracts (CMTATStandalone, CMTATUpgradeable,...) to inherits from the different modules

Name	Level	Description	Associated contracts deployments
CMTATBaseCommon	0	Inherits from all core and extension modules, except ValidationModule	No deployment contract directly inherits from this base contract (see next level)
CMTATBaseCore	0	Inherits from all core modules	CMTAT Light (Upgradeable & Standalone
CMTATBaseGeneric	0	Inherits from non-ERC20 related modules	- (Only mock available)
CMTATBaseAllowlist	1	Inherits from CMTATBaseCommon, but also from ValidationModuleAllowlist	CMTAT Allowlist (upgradeable & Standalone)
CMTATBaseRuleEngine	1	Add RuleEngine support by inheriting from ValidationModuleRuleEngine	No deployment contract directly inherits from this base contract (see next level)

Name	Level	Description	Associated contracts deployments
CMTATBaseDebt	2	Add debt support by inheriting from Debt and DebtEngine module	CMTAT Debt (Standalone & Upgradeable)
CMTATBaseERC1404	2	Add ERC-1404 support	CMTAT Standalone / Upgradeable
CMTATBaseERC20CrossChain	3	Add cross-chain support	No deployment contract directly inherits from this base contract (see next level)
CMTATBaseERC2771	4	Add ERC-2771 support by inheriting from ERC2771Module	CMTAT Standalone / Upgradeable CMTAT Upgradeable UUPS
CMTATBaseERC1363	5	Add ERC-1363 support by inheriting directly from OpenZeppelin contract	CMTAT ERC1363 (Upgradeable & Standalone)
CMTATBaseERC7551	5	Add ERC-7551 support by inheriting from ERC7551 Module	CMTAT ERC7551 (Upgradeable & Standalone)

Level 0 (main modules)

CMTAT Base Common

CMTAT Base adds several functions:

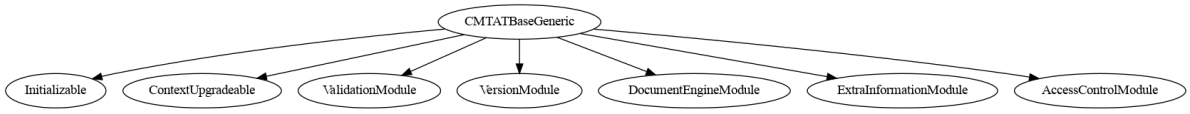
- `burnAndMint` to burn and mint atomically in the same function.

CMTAT Base Core

CMTAT Base Core adds several functions:

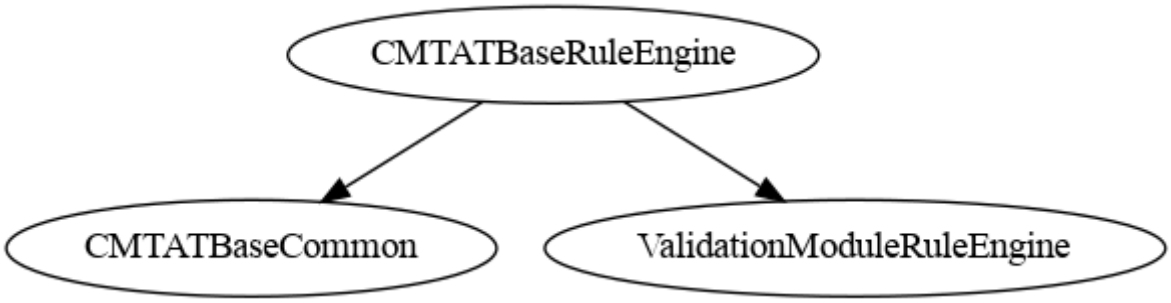
- `burnAndMint` to burn and mint atomically in the same function.
- `forcedBurn` to allow the admin to burn tokens from a frozen address (defined in EnforcementModule)
    - This function is not required in CMTATBase because the function `forcedTransfer` (ERC20EnforcementModule) can be used instead.

CMTAT Base Generic

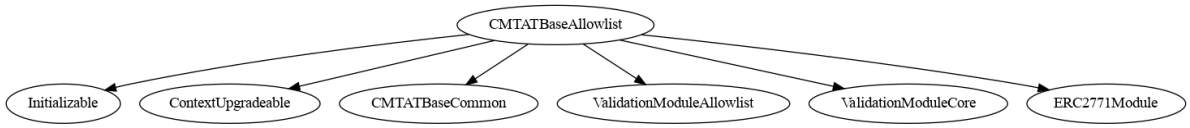


Level 1 (ERC-20 Transfer restriction)

CMTAT Base RuleEngine

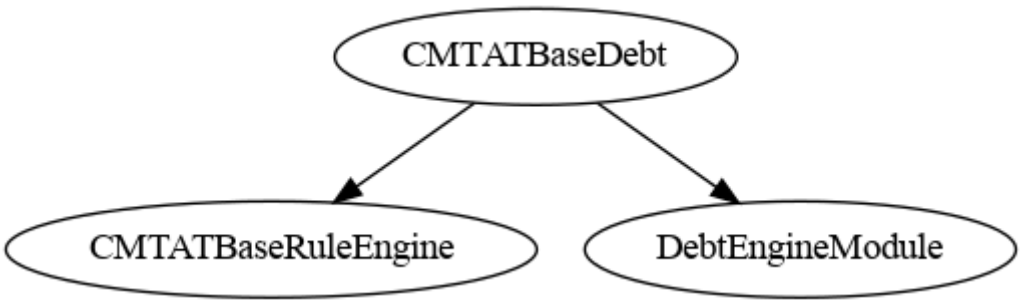


CMTAT Base Allowlist

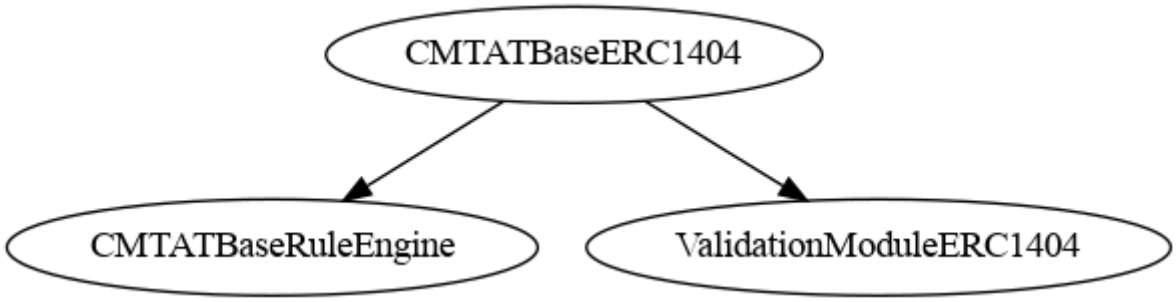


Level 2 (add heavy modules)

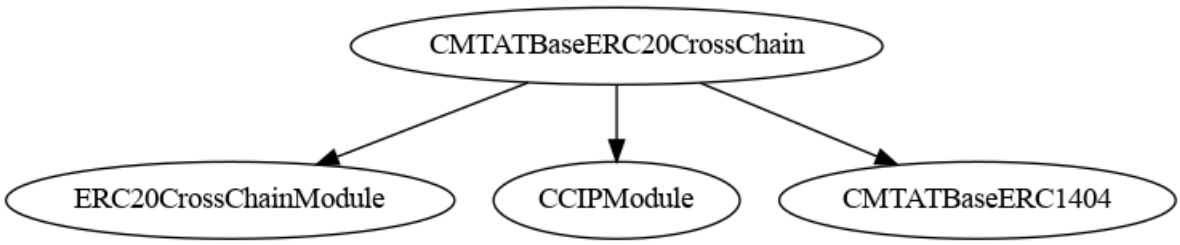
CMTATBaseDebt



CMTATBaseERC1404

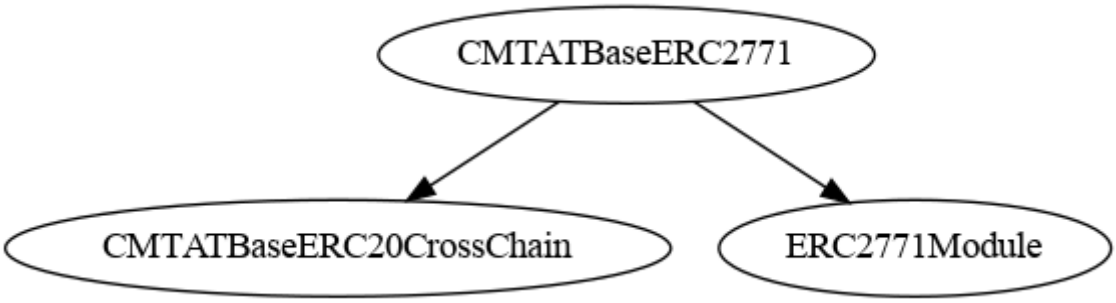


Level 3 (cross-chain transfer)



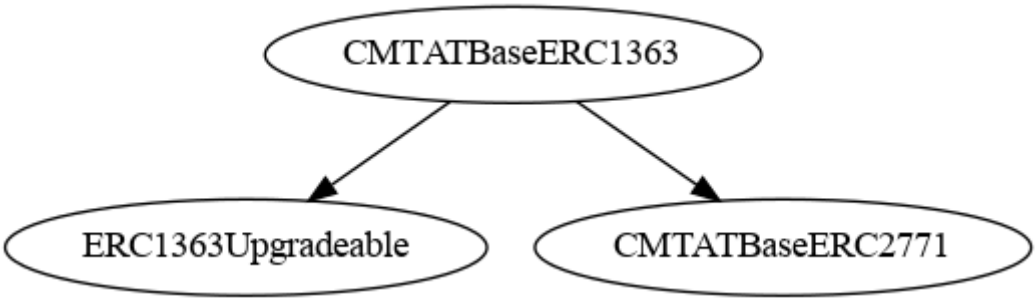
Level 4 (metaTx)

CMTAT Base ERC2771

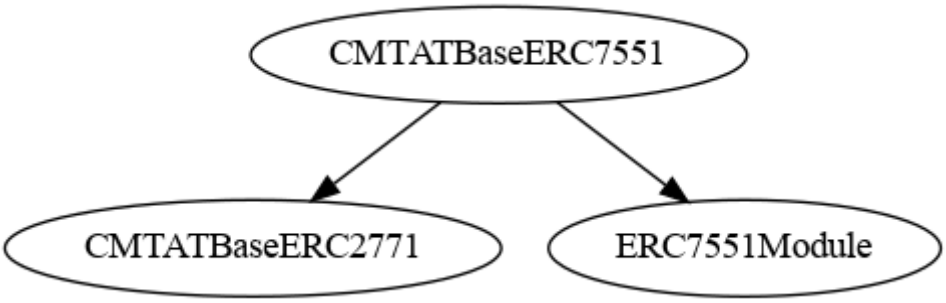


Level 5 (use case)

CMTAT Base ERC1363 (payable token)



CMTAT Base ERC7551



Module

Description

Modules describe a **logical** code separation inside CMTAT. They are defined as abstract contracts.
Their code and functionalities are part of the CMTAT and therefore are also included in the calculation of the contract size and the maximum size limit of 24 kB.

It is always possible to delete a module, but this requires modifying the code and compiling it again, which would require a security audit to be performed on these modifications.

Modules are also separated in different categories.

- **Internal** modules: implementation for a module when OpenZeppelin does not provide a library to use. For example, this is the case for the `EnforcementModule`.
- **Wrapper** modules: abstract contract around OpenZeppelin contracts or internal module. For example, the wrapper `PauseModule` provides public functions to call the internal functions from OpenZeppelin.
    - **Core** (Wrapper sub-category): Contains the modules required to be CMTA compliant
    - **Security**: module related to access control
    - **Extension** (Wrapper sub-category): not required to be CMTA compliant, "bonus features" (snapshotModule, debtModule)
    - **Options**: also bonus features to meet specific use cases through specific deployment version.

List

Here is the list of modules supported between different versions and the differences.

For simplicity, the module names and function locations are those of version 3.0.0

- "fn" means function
- Changes made in a release are considered maintained in the following release unless explicitly stated otherwise

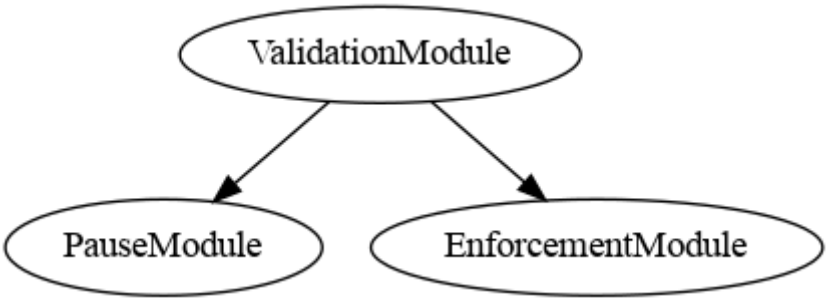
Controllers

Modules	Type	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT >= 3.0.0			
Deployment version						Standalone, Upgradeable, UUPS, Debt, ERC1363, ERC7551	CMTAT Debt	CMTAT Allowlist	CMTAT Light
ValidationModule	Controllers	Check transfer validity by calling the Pause and Enforcement modules	ValidationModule.sol	☑	☑	☑	☑	☑	☑
ValidationModuleAllowlist	Controllers	Check transfer validity by calling Allowlist module	ValidationModuleAllowlist.sol	☒	☒	☒	☒	☑	☒
ValidationModuleRuleEngineInternal	Internal	Configure a `RuleEngine`	ValidationModuleRuleEngineInternal.sol	☑	☑	☑	☑	☒	☒
ValidationModuleCore	Core	Implements `canTransfer` and `canTransferFrom` The core module does not implement ERC-1404 and the RuleEngine	ValidationModuleCore.sol	☑	☑	☑	☑	☑	☑

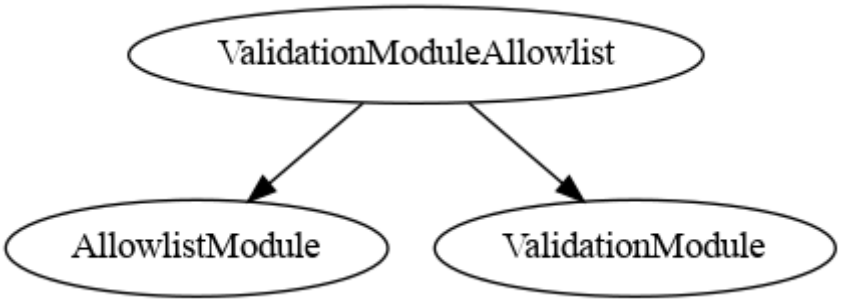
Modules	Type	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT >= 3.0.0			
ValidationModuleRuleEngine	Extensions	Set and call the ruleEngine to check transfer.	ValidationModuleRuleEngine.sol	☑	☑	☑	☑	☒	☒
ValidationModuleERC1404	Extensions	Implements ERC-1404	ValidationModuleERC1404.sol	☑	☑	☑	☒	☒	☒

Controllers

- ValidationModule

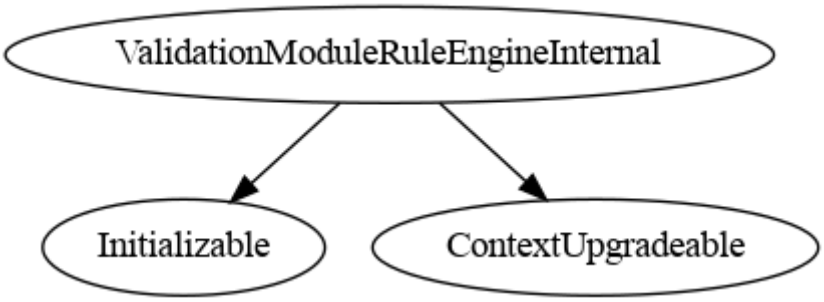


- ValidationModuleAllowlist



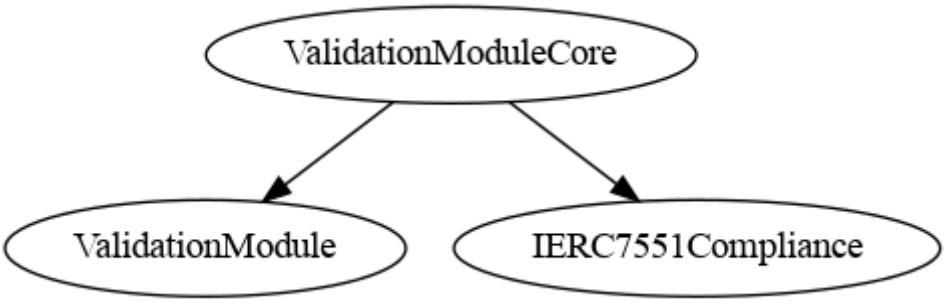
Internal

- ValidationModuleRuleEngineInternal



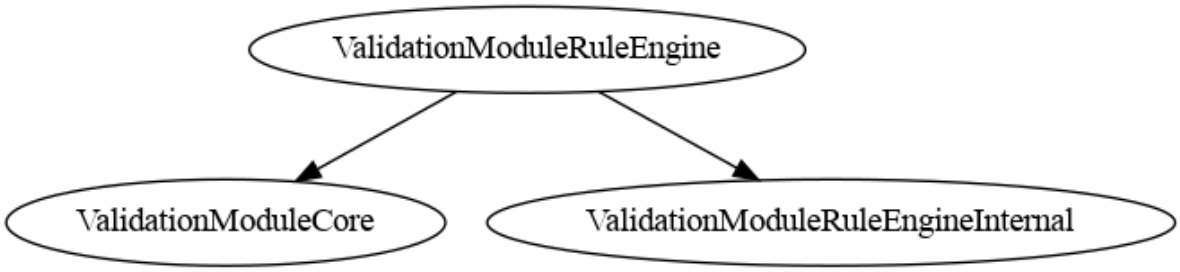
Core

- ValidationModuleCore

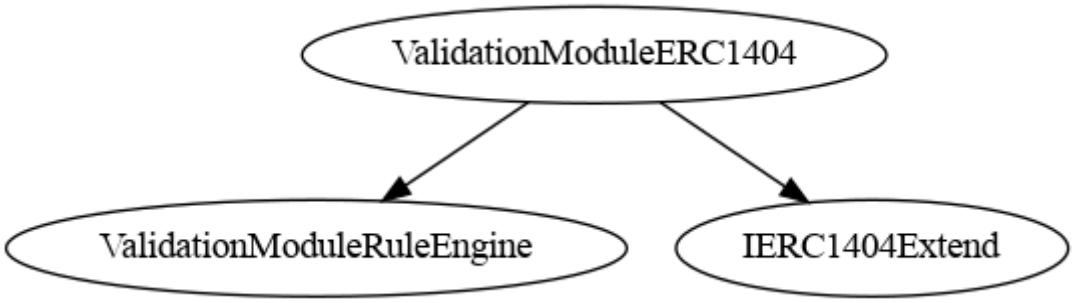


Extensions

- ValidationModuleRuleEngine



- ValidationModuleERC1404



Core modules

Generally, these modules are required to be compliant with the CMTA specification.

Modules	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT >= 3.0.0
VersionModule	Contract version	VersionModule.sol	☑	☑ (Add two fields: flag and information)	☑ Remove field flag (not used) Keep only the field VERSION and move the rest (tokenId, information,..) to an extension module `ExtraInformation` CMTAT 3.0.1: update name baseModule -> VersionModule
ERC20 Burn (Prev. BurnModule)	Burn functions	ERC20BurnModule.sol	☑	☑ Replace fn `burnFrom` by fn `forcedBurn`	Add fn `burnBatch` Rename `forceBurn` in `burn` `burnFrom` is moved to the option module `ERC20CrossChain` (v3.1.0)
Enforcement	Freeze/unfreeze address	EnforcementModule.sol	☑	☑	☑
ERC20Base	decimals, set name & symbol	ERC20BaseModule.sol	☑	☑ Remove fn `forceTransfer` (replaced by `burn` and `mint`)	Add fn `balanceInfo` (useful to distribute dividends) Add fn `forcedTransfer` Add fn `setName` and `setSymbol` Remove custom fn `approve` (keep only ERC-20 approve)
ERC20 Mint	Mint functions + BatchTransfer	ERC20MintModule.sol	☑	☑	Add fn `mintBatch` Add fn `transferBatch`
Pause Module	Pause and deactivate contract	PauseModule.sol	☑	☑	Replace fn `kill` by fn `deactivateContract`

Extensions modules

Generally, these modules are not required to be compliant with the CMTA specification.

Modules	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT >= 3.0.0
ExtraInformation	Set extra information (tokenId, terms, metadata)	ExtraInformationModule.sol	☑(BaseModule)	☑(BaseModule)	☑
SnapshotEngineModule (Prev. SnapshotModule)	Set snapshotEngine	SnapshotEngineModule.sol	☑	Partial (Not included by default because unaudited)	☑ (require an external SnapshotEngine)
DocumentEngineModule	Set additional document (ERC1643) through a DocumentEngine	DocumentEngineModule.sol	☒	☒	☑
ERC20EnforcementModule	The admin (or a third party appointed by it) can partially freeze a part of the balance of a token holder.	ERC20EnforcementModule.sol	☒	☒	☑

Options modules

Modules	Description	File	CMTAT 1.0	CMTAT 2.3.0	CMTAT >= 3.0.0			
Deployment version					Standalone & Upgradeable	Allowlist	Debt	ERC7551
DebtModule	Set Debt Info	DebtModule.sol	☒	☑	☒	☒	☑ (Don't include CreditEvents managed by DebtEngineModule)	☒
DebtEngineModule	Add a DebtEngine module (requires to set CreditEvents)	DebtEngineModule.sol	☒	☒	☒	☒	☑	☒
ERC2771Module	ERC-2771 support	ERC2771Module.sol	☑	☑ (forwarder immutable)	☑	☑	☒	☑
Allowlist	Add integrated allowlist support	AllowlistModule.sol	☒	☒	☒	☑	☒	☒
ERC7551Module	Add specific ERC-7551 functions	ERC7551Module.sol	☒	☒	☒	☒	☒	☑
ERC20CrossChainModule	Add cross-chain functionalities (ERC-7802, burn, burnFrom)	ERC20CrossChainModule.sol	☒	☒	☑ (v3.1.0)	☒	☒	☑
CCIPModule	Add CCIP specific function	CCIPModule.sol	☒	☒	☑ (v3.1.0)	☒	☒	☑

Security

	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT >= 3.0.0
AccessControlModule	Access Control	AccessControlModule.sol	☑	☑ (Admin has all the roles by default)	☑

Access Control (RBAC)

CMTAT access control is also modular and flexible.

Wrapper modules

Firstly, wrapper modules will separately:

- define the roles useful to restrict its own functions
- define virtual functions `authorize<specific role name>` which require to be overridden in CMTAT base module to add access control check.

To allow flexibility and customization, wrapper modules do not implement themselves the access control. Access control is defined in CMTAT base modules. Therefore, it is possible to create a new base module to use a different access control.

CMTAT base module

Current CMTAT base module use the standard RBAC access control by using the contract `AccessControl` from OpenZeppelin.

The `AccessControlModule` which is used by the different CMTAT base module and deployment contracts override the OpenZeppelin function `hasRole` to give by default all the roles to the `admin`.

See also docs.openzeppelin.com - AccessControl

Key management

As with any token contract, access to the admin key must be adequately restricted.

Likewise, access to the proxy contract must be restricted and segregated from the token contract.

UUPS Proxy

For the deployment version for **UUPS proxies**, unfortunately there is no segregation between contract rights (admin) and the proxy. A possible improvement would be to add an owner who would only have the rights to update the proxy.

Any compromise to the DEFAULT_ADMIN_ROLE account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Role list

Here is the list of roles and their 32 bytes identifier.

	Defined in	**32 bytes identifier**
DEFAULT_ADMIN_ROLE	OpenZeppelin AccessControl	0x0000000000000000000000000000000000000000000000000000000000000000
Core Modules		
BURNER_ROLE	BurnModule	0x3c11d16cbaffd01df69ce1c404f6340ee057498f5f00246190ea54220576a848
MINTER_ROLE	MintModule	0x9f2df0fed2c77648de5860a4cc508cd0818c85b8b8a1ab4ceeef8d981c8956a6
ENFORCER_ROLE	EnforcementModule	0x973ef39d76cc2c6090feab1c030bec6ab5db557f64df047a4c4f9b5953cf1df3
PAUSER_ROLE	PauseModule	0x65d7a28e3265b37a6474929f336521b332c1681b933f6cb9f3376673440d862a
Extension Modules		
SNAPSHOOTER_ROLE	SnashotModule	0x809a0fc49fc0600540f1d39e23454e1f6f215bc7505fa22b17c154616570ddef
DOCUMENT_ROLE	DocumentModule	0xdd7c9aafbb91d54fb2041db1d5b172ea665309b32f5fffdbddf452802a1e3b20

	Defined in	32 bytes identifier
EXTRA_INFORMATION_ROLE	ExtraInformationModule (Also used by ERC7551 module)	0x921df7a58eb4ea112afa962b8186161404ecda2e8fe97f8246026d02ad1a74b7
ERC20ENFORCER_ROLE	ERC20EnforcementModule	0xd62f75bf68b069bc8e2abd495a949fafec67a4e5a5b7cb36aedf0dd51eec7e72
Option Modules		
ALLOWLIST_ROLE	AllowlistModule	0x26a560d834a19637eccba4611bbc09fb32970bb627da0a70f14f83fdc9822cbc
DEBT_ROLE	DebtModule (also used by DebtEngineModule)	0xc6f3350ab30f55ce45863160fc345c1663d4633fe7cacfd3b9bbb6420a9147f8
CROSS_CHAIN_ROLE	ERC20CrossChainModule	0x620d362b92b6ef580d4e86c5675d679fe08d31dff47b72f281959a4eecdd036a
BURNER_FROM_ROLE	ERC20CrossChainModule	0x5bfe08abba057c54e6a28bce27ce8c53eb21d7a94376a70d475b5dee60b6c4e2
BURNER_SELF_ROLE	ERC20CrossChainModule	0x13d9f3ea33477b975af6cd01437366c28412d5bd9b872fa0fc25bd3a160683af

Role by modules

Here a summary tab for each restricted functions defined in a module
For function signatures, struct arguments are represented with their corresponding native type.

Roles are defined in their specific modules but enforced in CMTAT Base module.

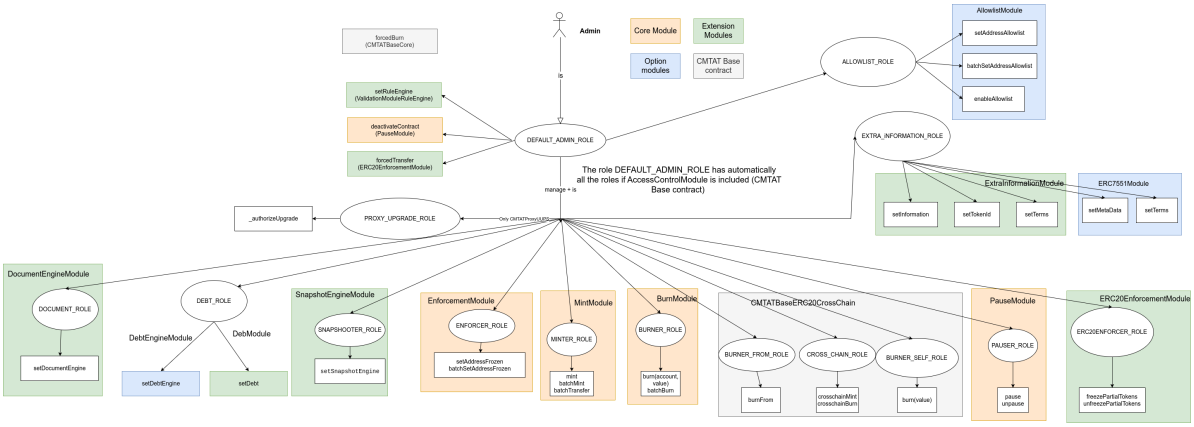
Thus, you are free to use a module, for example `PauseModule` and apply a different access control to restrict the function.

	Function signature	Visibility [public/external]	Input variables (Function arguments)	Output variables (return value)	Role Required
Core Modules					
ERC20BaseModule					
	`setName(string name_)`	public	`string name_`	-	DEFAULT_ADMIN_ROLE
	`setSymbol(string symbol_)`	public	`string symbol_`	-	DEFAULT_ADMIN_ROLE
ERC20BurnModule					
	`burn(address account, uint256 value, bytes data)`	public	`address account, uint256 value, bytes data`	-	BURNER_ROLE
	`burn(address account, uint256 value)`	public	`address account, uint256 value`	-	BURNER_ROLE
	`batchBurn(address[] accounts, uint256[] values, bytes data)`	public	`address[] accounts, uint256[] values, bytes data`	-	BURNER_ROLE
	`batchBurn(address[] accounts, uint256[] values)`	public	`address[] accounts, uint256[] values`	-	BURNER_ROLE
ERC20MintModule					
	`mint(address account, uint256 value, bytes data)`	public	`address account, uint256 value, bytes data`	-	MINTER_ROLE
	`mint(address account, uint256 value)`	public	`address account, uint256 value`	-	MINTER_ROLE
	`batchMint(address[] accounts, uint256[] values)`	public	`address[] accounts, uint256[] values`	-	MINTER_ROLE
	`batchTransfer( address[] tos, uint256[] values)`	public	`address[] tos, uint256[] values`	bool	MINTER_ROLE
EnforcementModule					
	`setAddressFrozen(address account, bool freeze)`	public	`address account, bool freeze`	-	ENFORCER_ROLE
	`setAddressFrozen(address account, bool freeze, bytes data)`	public	`address account, bool freeze, bytes data`	-	ENFORCER_ROLE
	`batchSetAddressFrozen( address[] accounts, bool[] freezes)`	public	`address[] accounts, bool[] freezes`	-	ENFORCER_ROLE
PauseModule					
	`pause()`	public	-	-	PAUSER_ROLE
	`unpause()`	public	-	-	PAUSER_ROLE
	`deactivateContract()`	public	-	-	DEFAULT_ADMIN_ROLE
Extension Modules					
DocumentEngineModule					
	`setDocumentEngine(address documentEngine_ )`	public	`IERC1643 documentEngine_`	-	DOCUMENT_ROLE
ERC20EnforcementModule					
	`forcedTransfer(address from, address to, uint256 value, bytes data)`	public	`address from, address to, uint256 value, bytes data`	bool	DEFAULT_ADMIN_ROLE
	`forcedTransfer(address from, address to, uint256 value)`	public	`address from, address to, uint256 value`	bool	DEFAULT_ADMIN_ROLE
	`freezePartialTokens(address account, uint256 value)`	public	`address account, uint256 value`	-	ERC20ENFORCER_ROLE
	`unfreezePartialTokens(address account, uint256 value)`	public	`address account, uint256 value`	-	ERC20ENFORCER_ROLE
	`freezePartialTokens(address account, uint256 value, bytes data)`	public	`address account, uint256 value, bytes data`	-	ERC20ENFORCER_ROLE
	`unfreezePartialTokens(address account, uint256 value, bytes data)`	public	`address account, uint256 value, bytes data`	-	ERC20ENFORCER_ROLE
ExtraInformationModule					
	`setTokenId(string tokenId_ )`	public			EXTRA_INFORMATION_ROLE
	`setTerms((string,string,bytes32) terms_)`	public	`IERC1643CMTAT.DocumentInfo terms_`		
	`setInformation(string information_)`	public	`string information_`		
SnapshotEngineModule					ERC20ENFORCER_ROLE
	`setSnapshotEngine(address snapshotEngine_)`	public	`ISnapshotEngine snapshotEngine_`		SNAPSHOOTER_ROLE
Option Modules					
AllowlistModule					

	Function signature	Visibility [public/external]	Input variables (Function arguments)	Output variables (return value)	Role Required
	`setAddressAllowlist(address account, bool status)`	public	`address account, bool status`	-	ALLOWLIST_ROLE
	`setAddressAllowlist(address account, bool status, bytes data)`	public	`address account, bool status, bytes data`	-	ALLOWLIST_ROLE
	`batchSetAddressAllowlist(address[] accounts, bool[] status)`	public	`address[] accounts, bool[] status`	-	ALLOWLIST_ROLE
DebtEngineModule					BURNER_FROM_ROLE
	`setDebtEngine(address debtEngine_)`	public	`IDebtEngine debtEngine_`	-	DEBT_ROLE
DebtModule					
	`setCreditEvents( (bool,bool,string) creditEvents_)`	public	`CreditEvents creditEvents_`	-	DEBT_ROLE
	`setDebt((string,string,string,string),(uint256,uint256,uint256,string,string,string,string,string,string,string,address) debt_)`	public	`ICMTATDebt.DebtInformation debt_`	-	DEBT_ROLE
ERC7551Module					
	`setMetaData(string metadata_)`	public	`string metadata_`	-	EXTRA_INFORMATION_ROLE
	`setTerms(bytes32 hash, string uri)`	public	`bytes32 hash, string uri`	-	EXTRA_INFORMATION_ROLE
ERC20CrossChain					
	`crosschainMint(address to, uint256 value)`	public	`address to, uint256 value`	-	CROSS_CHAIN_ROLE
	`crosschainBurn(address from, uint256 value)`	public	`address from, uint256 value`	-	CROSS_CHAIN_ROLE
	`burnFrom(address account, uint256 value)`	public	`address account, uint256 value`	-	BURNER_FROM_ROLE
	`burn(uint256 value)`	public	`uint256 value`	-	BURNER_SELF_ROLE
CCIPModule					
	`setCCIPAdmin(address newAdmin)`	public	`address newAdmin`	-	DEFAULT_ADMIN_ROLE
Base contract					
BaseCommon					
	`burnAndMint(address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes data)`	public	`address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes data`	-	Same role requirement as `burn` and `mint`, so BURNER_ROLE and MINTER_ROLE
CMTATBaseCore (only CMTAT light version)					
	`burnAndMint(address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes data)`	public	`address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes data`	-	Same role requirement as `burn` and `mint`, so BURNER_ROLE and MINTER_ROLE
	`forcedBurn(address account, uint256 value, bytes data)`	public	`address account, uint256 value, bytes data`	-	DEFAULT_ADMIN_ROLE

Schema

This schema contains the different roles and their restricted functions.



The OpenZepplin functions `grantRole` and `revokeRole` can be used by the admin to grant and revoke role to an address.

Transfer adminship

To transfer the adminship to a new admin, the current admin must call two functions:

1. `grantRole()` by specifying the DEFAULT_ADMIN_ROLE identifier and the new admin address
2. `renounceRole()` to revoke the DEFAULT_ADMIN_ROLE from its own account.

The new admin can also revoke a role from the current/old admin by calling `revokeRole`.

It is also possible to have several different admins.

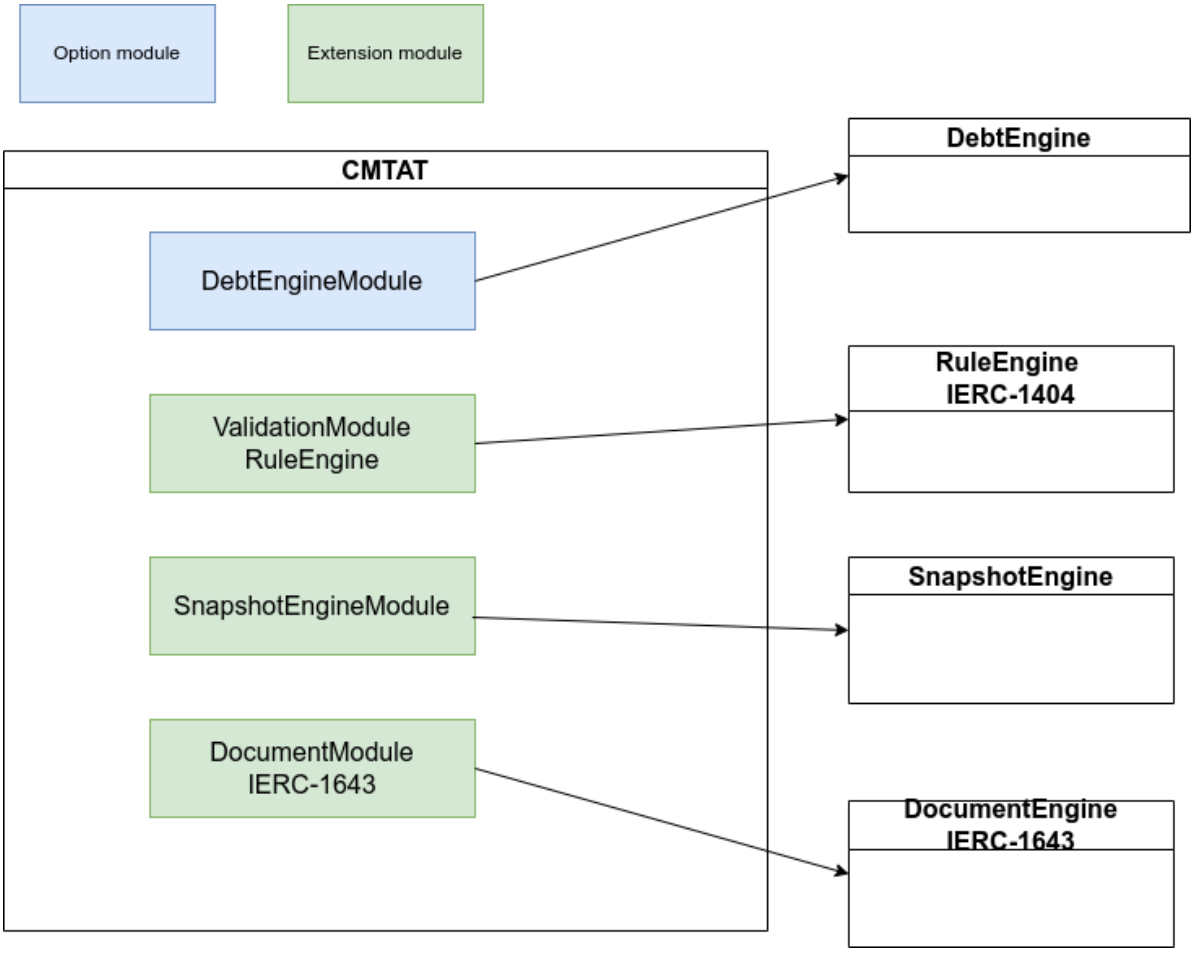
Engines

Engines are external smart contracts called by CMTAT modules.

These engines are **optional** and their addresses can be left to zero.

Schema

Here is a schema with the different modules and the associated engines.



RuleEngine (IERC-1404)

The `RuleEngine` is an external contract used to apply transfer restrictions to the CMTAT through whitelisting, blacklisting,...

This contract is defined in the `ValidationModule`.

Requirement

Since the version v3.0.0, the requirements to use a RuleEngine are the following:

> The `RuleEngine` must import and implement the interface `IRuleEngine` which declares the ERC-3643 functions `transferred`(read-write) and `canTransfer`(ready-only) with several other functions related to ERC-1404, ERC-7551 and ERC-3643.

This interface can be found in IRuleEngine.sol.

Warning: The `RuleEngine` has to restrict the access of the function `transferred` to only the `CMTAT token contract`.

How it works

Before each transfer (standard transfer/mint/burn), the CMTAT calls the ERC-3643 function `transferred` which is the entrypoint for the RuleEngine.

```
function transferred(address from, address to, uint256 value) external;
```

CMTAT defines the interaction with the RuleEngine inside a specific module, ValidationModuleRuleEngine and CMTATBaseRuleEngine.

- ValidationModuleRuleEngine

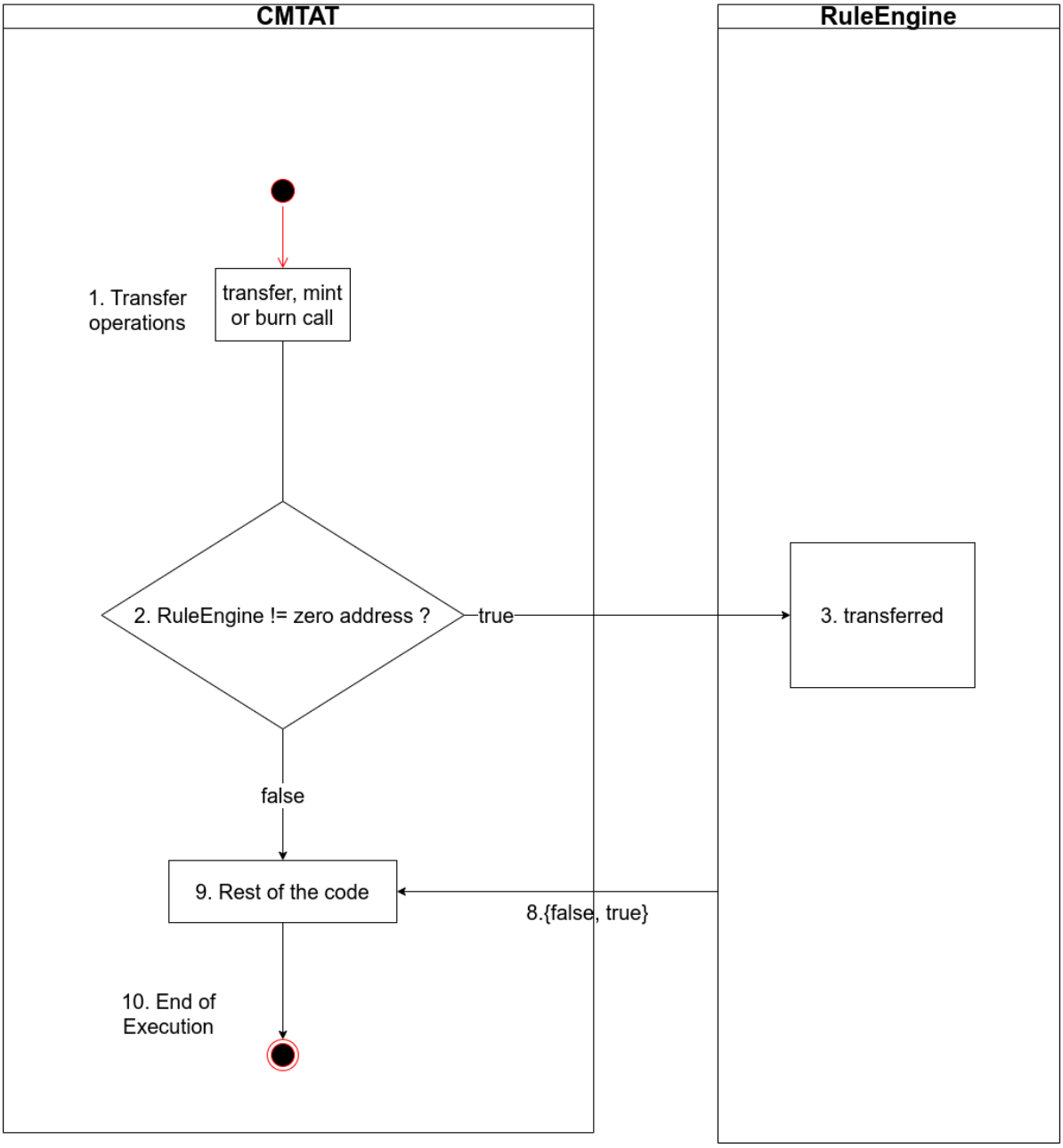
```
/* ============ State functions ============ */
function _transferred(address spender, address from, address to, uint256 value) internal virtual returns (bool){
    if(!_canTransferGenericByModule(spender, from, to)){
        return false;
    } else {
         IRuleEngine ruleEngine_ = ruleEngine();
         if (address(ruleEngine_) != address(0)){
             if(spender != address(0)){
                ruleEngine_.transferred(spender, from, to, value);
             } else {
                ruleEngine_.transferred(from, to, value);
             }
        }
    }
    return true;
}
```

- CMTATBaseRuleEngine

```
function _checkTransferred(address spender, address from, address to, uint256 value) internal virtual override(CMTATBaseCommon) {
    CMTATBaseCommon._checkTransferred(spender, from, to, value);
    require(ValidationModuleRuleEngine._transferred(spender, from, to, value), Errors.CMTAT_InvalidTransfer(from, to, value));
}
```

This function `_transferred` is called before each transfer/burn/mint through the internal function `_checkTransferred` defined in CMTAT_BASE.

Here is a schema to show how it works:



1. The token holders initiate a transfer transaction on CMTAT contract.
2. The validation module inside the CMTAT calls the ERC-3643 function `transferred` from the RuleEngine if set with the following parameters inside: `from, to, value`.
3. The Rule Engine performs the restriction check and revert if the transfer is not authorised.

TransferFrom - Spender restriction

The `RuleEngine` is also called with the function `transferFrom`.

In this case, the `transferred` function called contains an additional `spender` argument:

```
function transferred(address spender, address from, address to, uint256 value)
external;
```

This allows the `RuleEngine` to also apply restriction on the spender.

Interface

Here the list of functions defined by the RuleEngine interface through inheritance

```
// IRuleEngine
function transferred(address spender, address from, address to, uint256 value)
external;

// IERC-1404
function detectTransferRestriction(address from,address to,uint256 value)
external view returns (uint8);

function messageForTransferRestriction(uint8 restrictionCode)
external view returns (string memory);

// IERC-1404Extend
enum REJECTED_CODE_BASE {
        TRANSFER_OK,
        TRANSFER_REJECTED_DEACTIVATED,
        TRANSFER_REJECTED_PAUSED,
        TRANSFER_REJECTED_FROM_FROZEN,
        TRANSFER_REJECTED_TO_FROZEN,
        TRANSFER_REJECTED_SPENDER_FROZEN,
        TRANSFER_REJECTED_FROM_INSUFFICIENT_ACTIVE_BALANCE
    }

function detectTransferRestrictionFrom(address spender,address from,address
to,uint256 value)
external view returns (uint8);


// IERC7551Compliance
function canTransferFrom(address spender,address from,address to,uint256 value)
external view returns (bool);


// IER3643ComplianceRead
function canTransfer(address from,address to,uint256 value)
external view returns (bool isValid);

// IERC3643IComplianceContract
function transferred(address from, address to, uint256 value)
external;
```

Interface details

IRuleEngine

`IRuleEngine` is the main interface which inherits from all other interfaces: `IERC1404Extend`, `IERC7551Compliance` and `IERC3643IComplianceContract`.

```
interface IRuleEngine is IERC1404Extend, IERC7551Compliance,
IERC3643IComplianceContract {
```

```
    /**
     *  @notice
     *  Function called whenever tokens are transferred from one wallet to
another
     *  @dev
     *  Must revert if the transfer is invalid
     *  Same name as ERC-3643 but with one supplementary argument `spender`
     *  This function can be used to update state variables of the RuleEngine
contract
     *  This function can be called ONLY by the token contract bound to the
RuleEngine
     *  @param spender spender address (sender)
     *  @param from token holder address
     *  @param to receiver address
     *  @param value value of tokens involved in the transfer
     */
    function transferred(address spender, address from, address to, uint256
value) external;
}
```

IERC7551 & ERC-3643 Compliance

A RuleEngine must implement the ERC-7551 function `canTransferFrom` & ERC-3643 compliance function `canTransfer`.

```
interface IERC7551Compliance is IERC3643ComplianceRead {
    /*
    * @notice This function return true if the message sender is able to
transfer amount tokens to to respecting all compliance.
    * @dev Don't check the balance and the user's right (access control)
    */
    function canTransferFrom(
        address spender,
        address from,
        address to,
        uint256 value
    )  external view returns (bool);
}
interface IERC3643ComplianceRead {
    /**
     * @notice Returns true if the transfer is valid, and false otherwise.
     * @dev Don't check the balance and the user's right (access control)
     */
    function canTransfer(
        address from,
        address to,
        uint256 value
    ) external view returns (bool isValid);
}
```

ERC-1404 & ERC1404Extend

A RuleEngine must implement the `ERC1404Extend` interface which inherits from `IERC1404`

- IERC1404

```
interface IERC1404 {

    /**
     * @notice Returns a uint8 code to indicate if a transfer is restricted or
not
     * @dev
     * See {ERC-1404}
     * This function is where an issuer enforces the restriction logic of their
token transfers.
     * Some examples of this might include:
     * - checking if the token recipient is whitelisted,
     * - checking if a sender's tokens are frozen in a lock-up period, etc.
     * @return uint8 restricted code, 0 means the transfer is authorized
     *
     */
    function detectTransferRestriction(
        address from,
        address to,
        uint256 value
    ) external view returns (uint8);


    /**
     * @dev See {ERC-1404}
     * This function is effectively an accessor for the "message",
     * a human-readable explanation as to why a transaction is restricted.
     *
     */
    function messageForTransferRestriction(
        uint8 restrictionCode
    ) external view returns (string memory);
}
```

- IERC1404Extend

```
/**
* @title IERC1404 with custom related extensions
*/
interface IERC1404Extend is IERC1404{
    /*
    * @dev leave the code 6-9 free/unused for further CMTAT additions in your
ruleEngine implementation
    */
    enum REJECTED_CODE_BASE {
        TRANSFER_OK,
        TRANSFER_REJECTED_PAUSED,
        TRANSFER_REJECTED_FROM_FROZEN,
```

```
        TRANSFER_REJECTED_TO_FROZEN,
        TRANSFER_REJECTED_SPENDER_FROZEN,
        TRANSFER_REJECTED_FROM_INSUFFICIENT_ACTIVE_BALANCE
    }

    /**
     * @notice Returns a uint8 code to indicate if a transfer is restricted or
not
     * @dev
     * See {ERC-1404}
     * Add an additionnal argument `spender`
     * This function is where an issuer enforces the restriction logic of their
token transfers.
     * Some examples of this might include:
     * - checking if the token recipient is whitelisted,
     * - checking if a sender's tokens are frozen in a lock-up period, etc.
     * @return uint8 restricted code, 0 means the transfer is authorized
     *
     */
    function detectTransferRestrictionFrom(
        address spender,
        address from,
        address to,
        uint256 value
    ) external view returns (uint8);
}
```

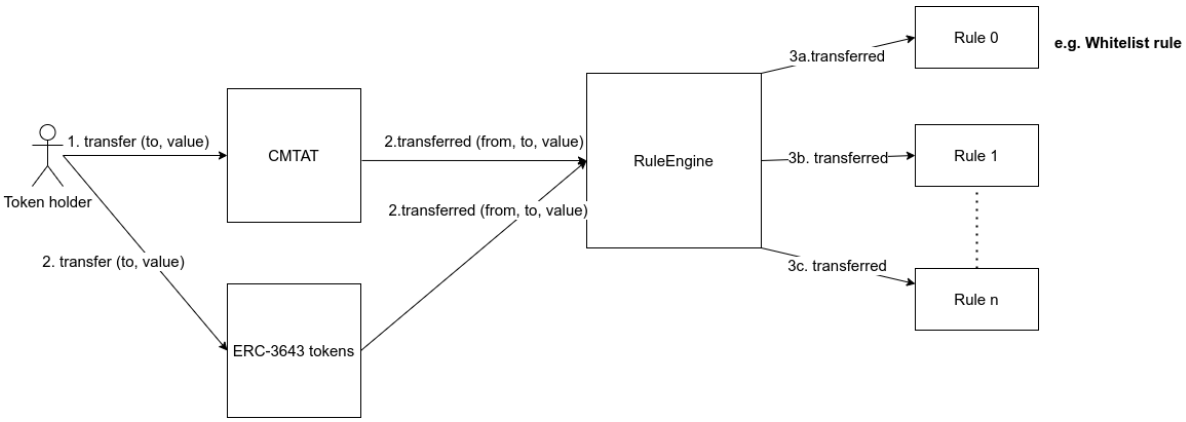
RuleEngine CMTA implementation

CMTA provides an implementation of a RuleEngine compatible with CMTAT. This RuleEngine is also compatible with ERC-3643 tokens.

In this implementation, the token holder calls the ERC-20 function `transfer` which triggers a call to the `RuleEngine` (ERC-3643 `transferred`) and the different rules associated.

The different rules are not included in the RuleEngine interface and you are free to build a different RuleEngine.

Schema



Version

Here is the list of the different versions available for each CMTAT version.

CMTAT version	RuleEngine
CMTAT v3.0.0	RuleEngine v3.0.0-rc0 (unaudited)
CMTAT 2.5.0 (unaudited)	RuleEngine >= v2.0.3 (unaudited)
CMTAT 2.4.0 (unaudited)	RuleEngine >=v2.0.0 Last version: v2.0.2(unaudited)
CMTAT 2.3.0	RuleEngine v1.0.2
CMTAT 2.0 (unaudited)	RuleEngine 1.0 (unaudited)
CMTAT 1.0	No ruleEngine available

This contract acts as a controller and can call different contract rules to apply rules on each transfer.

Rules

Rules have their own dedicated repository: github.com/CMTA/Rules and they can be used through a RuleEngine or directly with CMTAT.

Here are the list of rules in development:

Rule	Type [ready-only / read-write]	Security Audit planned in the roadmap	Description
RuleWhitelist	Ready-only	☑	This rule can be used to restrict transfers from/to only addresses inside a whitelist.
RuleWhitelistWrapper	Ready-only	☑	This rule can be used to restrict transfers from/to only addresses inside a group of whitelist rules managed by different operators.
RuleBlacklist	Ready-only	☑	This rule can be used to forbid transfer from/to addresses in the blacklist

Rule	Type [ready-only / read-write]	Security Audit planned in the roadmap	Description
RuleSanctionList	Ready-only	☑	The purpose of this contract is to use the oracle contract from Chainalysis to forbid transfer from/to an address included in a sanctions designation (US, EU, or UN).
RuleConditionalTransferLight	Ready-Write	In development	This rule requires that transfers have to be approved before being executed by the token
RuleConditionalTransfer	Ready-Write	☒ (experimental rule)	Same principle as the light version (see above) but we more options such as a time limit for approving a request as well as for carrying out the transfer

SnapshotEngine

This Engine allows to perform snapshot on-chain.

- This engine is defined in the module `SnapshotModule`.
- CMTAT implements only one function defined in the interface ISnapshotEngine

Before each transfer, the CMTAT calls the function `operateOnTransfer` which is the entrypoint for the SnapshotEngine.

```
/*
* @dev minimum interface to define a SnapshotEngine
*/
interface ISnapshotEngine {
   /**
    * @notice Records balance and total supply snapshots before any token
transfer occurs.
    * @dev This function should be called inside the {_update} hook so that
    * snapshots are updated prior to any state changes from {_mint}, {_burn},
or {_transfer}.
    * It ensures historical balances and total supply remain accurate for
snapshot queries.
    *
    * @param from The address tokens are being transferred from (zero address
if minting).
    * @param to The address tokens are being transferred to (zero address if
burning).
```

```
    * @param balanceFrom The current balance of `from` before the transfer
(used to update snapshot).
    * @param balanceTo The current balance of `to` before the transfer (used to
update snapshot).
    * @param totalSupply The current total supply before the transfer (used to
update snapshot).
    */
    function operateOnTransfer(address from, address to, uint256 balanceFrom,
uint256 balanceTo, uint256 totalSupply) external;
}
```

SnapshotEngine CMTA implementation

CMTA provides an implementation of a SnapshotEngine compatible with CMTAT.

CMTAT	SnapshotEngine
CMTAT v3.0.0	v0.3.0 (unaudited)
CMTAT v2.3.0	SnapshotEngine v0.1.0 (unaudited)
CMTAT v2.4.0, v2.5.0 (unaudited)	Include inside SnapshotModule (unaudited)
CMTAT v2.3.0	Include inside SnapshotModule (unaudited)
CMTAT v1.0.0	Include inside SnapshotModule, but not gas efficient (audited)

CMTAT Snapshot - Deployment version

Instead of an external contract, it is also possible to extend CMTAT to include the logic to perform snapshots.

The SnapshotEngine repository provides also a specific deployment version which extends CMTAT to include a part of the SnapshotEngine codebase to perform snapshot on-chain.

DebtEngine

This engine can be used to configure Debt and Credits Events information

- It is defined in the `DebtEngineModule` (option module)
- It extends the `DebtModule` (option module) by allowing to set Credit Events and Debt info through an external contract called `DebtEngine`.
- If a `DebtEngine` is configured, the function `debt` will return the debt configured by the `DebtEngine` instead of the `DebtModule`.

This module only implements two functions, available in the interface IDebtEngine to get information from the `DebtEngine`.

```
interface IDebtEngine is ICMTATDebt, ICMTATCreditEvents {
    // nothing more
}
interface ICMTATDebt {
    /**
```

```
     * @notice Returns debt information
     */
    function debt() external view returns(DebtInformation memory);
}
interface ICMTATCreditEvents {
     /**
     * @notice Returns credit events
     */
    function creditEvents() external view returns(CreditEvents memory);
}
```

Use an external contract provides two advantages:

- Reduce code size of CMTAT, which is near of the maximal size limit
- Allow to manage this information for several different tokens (CMTAT or not).

Here is the list of the different version available for each CMTAT version.

CMTAT version	DebtEngine
CMTAT v3.0.0	Under development
CMTAT v2.5.0 (unaudited)	DebtEngine v0.2.0 (unaudited)

DocumentEngine (IERC-1643)

The `DocumentEngine` is an external contract to support *ERC-1643* inside CMTAT, a standard proposition to manage documents on-chain. This standard is notably used by ERC-1400 from Polymath.

This engine is defined in the module `DocumentModule`

This EIP defines a document with three attributes:

- A short name (represented as a `bytes32`)
    - In CMTAT, since this EIP is not official, we decided to use the type `string` instead of `bytes32` to allow `name` with more than 32 characters as suggested in this comment.
- A generic URI (represented as a `string`) that could point to a website or other document portal.
- The hash of the document contents associated with it on-chain.

CMTAT only implements two functions from this standard, available in the interface IERC1643 to get the documents from the documentEngine.

```
interface IERC1643 {
    struct Document {
        string uri;
        bytes32 documentHash;
        uint256 lastModified;
    }
    /**
     * @notice return a document identified by its name
     */
    function getDocument(string memory name) external view returns (Document
memory doc);
```

```
    /**
     * @notice return all documents
     */
    function getAllDocuments() external view returns (string[] memory);
}
```

The `DocumentEngine` has to import and implement this interface. To manage the documents, the engine is completely free on how to do it.

Using an external contract provides two advantages:

- Reduce code size of CMTAT, which is near the maximal size limit
- Allow documents management for several different tokens (CMTAT or not).

Here is the list of the different versions available for each CMTAT version.

CMTAT version	DocumentEngine
CMTAT v3.0.0	Under development
CMTAT v2.5.0 (unaudited)	DocumentEngine v0.3.0 (unaudited)

AuthorizationEngine (Deprecated)

> Warning: this engine has been removed since CMTAT v3.0.0

The `AuthorizationEngine` was an external contract to add supplementary checks on AccessControl (functions `grantRole` and `revokeRole`) from the CMTAT. Since delegating access rights to an external contract is complicated and it is better to manage access control directly in CMTAT, we removed it in version 3.0.0.

There was only one prototype available: CMTA/AuthorizationEngine

CMTAT version	AuthorizationEngine
CMTAT v3.0.0	Removed
CMTAT v2.4.0, 2.5.0, 2.5.1 (unaudited)	AuthorizationEngine v1.0.0 (unaudited)
CMTAT 2.3.0 (audited)	Not available
CMTAT 1.0 (audited)	Not available

Functionality details

ERC-20 properties

All ERC-20 properties (`name`, `symbol` and `decimals`) can be set at deployment or initialization if a proxy is used.

Once the contract is deployed, the core module `ERC20BaseModule` offers two ERC-3643 functions which allow to update the name and the symbol (but not the decimals).

```
interface IERC3643ERC20Base {
    /**
     *  @notice sets the token name
     */
    function setName(string calldata name) external;
    /**
     *  @notice sets the token symbol
     */
    function setSymbol(string calldata symbol) external;
}
```

MetaTx/Gasless support (ERC-2771 module)

The CMTAT supports client-side gasless transactions using the standard ERC-2771.

The contract uses the OpenZeppelin contract `ERC2771ContextUpgradeable`, which allows a contract to get the original client with `_msgSender()` instead of the feepayer given by `msg.sender`.

At deployment, the parameter `forwarder` inside the CMTAT contract constructor has to be set with the defined address of the forwarder.

After deployment:

- In standalone deployment, the forwarder is immutable and can not be changed after deployment.
- In upgradeable deployment (with a proxy), it is possible to change the forwarder by deploying a new implementation. This is possible because the forwarder is stored inside the implementation contract bytecode instead of the proxy's storage.

References:

- OpenZeppelin Meta Transactions
- OpenGSN has deployed several forwarders, see their documentation to see some examples.

Enforcement / Transfer restriction

There are several ways to restrict transfers as well as burn/mint operations.

Enforcement Module

Specific addresses can be frozen with the following ERC-3643 functions `setAddressFrozen` and `batchSetAddressFrozen`

```
interface IERC3643Enforcement {
    function isFrozen(address account) external view returns (bool);
    function setAddressFrozen(address account, bool freeze) external;
    function batchSetAddressFrozen(address[] calldata accounts, bool[] calldata
freeze) external;
}
```

Additionally, a `data` parameter can be also used, which will be emitted inside the smart contract

```
function setAddressFrozen(address account, bool freeze, bytes calldata data)
```

Due to a limited contract size, there is no batch version with a data parameter available.

When an address is frozen, it is not possible to mint tokens to this address or burn its tokens. To move tokens from a frozen address, the issuer must use the function `forcedTransfer`.

ERC20EnforcementModule

- A part of the balance of a specific address can be frozen with the following ERC3643 function `freezePartialTokens` and `unfreezePartialTokens`
- Transfer/burn can be forced by the admin (ERC20EnforcementModule) with the following ERC3643 function `forcedTransfer`.
    - In this case, if a part of the balance is frozen, the tokens are unfrozen before being burnt or transferred.

```
interface IERC3643ERC20Enforcement {
    /**
     *  @notice Returns the amount of tokens that are partially frozen on a
wallet
     */
    function getFrozenTokens(address account) external view returns (uint256);

    /**
     *  @notice freezes token amount specified for given address.
     */
    function freezePartialTokens(address account, uint256 value) external;
    /**
     *  @notice unfreezes token amount specified for given address
     */
    function unfreezePartialTokens(address account, uint256 value) external;
    /**
     *
     *  @notice Triggers a forced transfer.
     */
    function forcedTransfer(address from, address to, uint256 value) external
returns (bool);
}
```

Pause & Deactivate contract (PauseModule)

Pause

- Standard transfers can be put in pause with the following ERC3643 function `pause` and `unpause`
- From ERC-3643

```
interface IERC3643Pause {
    /**
     * @notice Returns true if the contract is paused, and false otherwise.
     */
    function paused() external view returns (bool);
    /**
```

```
     *  @notice pauses the token contract,
     *  @dev When contract is paused token holders cannot transfer tokens
anymore
     *
     */
    function pause() external;

    /**
     *  @notice unpauses the token contract,
     *  @dev When contract is unpaused token holders can transfer tokens
     *
     */
    function unpause() external;
}
```

Note:

The pause function does not affect burn and mint operations implemented in the contracts `ERC20MintModule` and `ERC20BurnModule`.

By separating burn/mint from standard transfer, the admin can re-adjust the supply while the standard transfers are paused. The alternative in this case to block mint and burn operations is to remove the MINTER and BURNER roles from the addresses concerned.

On the other hand, specific function for cross-chain bridge (`3_CMTATBaseERC20CrossChain.sol`) will revert if contract is paused because they are not intended to be used by the issuer to manage the supply.

Future possible improvement:

An alternative solution would be to provide an additional function `pauseAllTransfers` which would pause standard transfers, as well as all burn and mint operations.
However, due to the architecture of current contracts, it is not possible to add this functionality without exceeding the maximum contract size on Ethereum.
Consideration will be given to how this can be achieved in a future release.

Deactivate contracts

```
interface ICMTATDeactivate {
    event Deactivated(address account);
    /**
    * @notice deactivate the contract
    * Warning: the operation is irreversible, be careful
    */
    function deactivateContract() external;

    /**
    * @notice Returns true if the contract is deactivated, and false otherwise.
    */
    function deactivated() external view returns (bool) ;
}
```

Since the version v2.3.1, a function `deactivateContract` is implemented in the PauseModule to deactivate the contract.

If a contract is deactivated, it is no longer possible to perform transfer and burn/mint operations.

Kill (previous version)

CMTAT initially supported a `kill()` function relying on the SELFDESTRUCT opcode (which effectively destroyed the contract's storage and code).
However, Ethereum's Cancun upgrade (rolled out in Q1 of 2024) has removed support for SELFDESTRUCT (see EIP-6780).

From then on, the `kill` function no longer worked as expected, and we have replaced it by the function `deactivateContract` .

How it works

Firstly, the contract must be in `pause` state, by calling the function `pause`, otherwise the function reverts.

This function sets a boolean state variable `isDeactivated` to true.
The function `unpause` is updated to revert if the previous variable is set to true, thus the contract is in the pause state "forever".

The consequences are the following:

- In standalone deployment, this operation is irreversible, it is not possible to rollback.
- In upgradeable deployment (with a proxy), it is still possible to rollback by deploying a new implementation which sets the variable `isDeactivated` to false.

Supply management (burn & mint)

This tab summarizes the different behavior of burn/mint functions if:

- The target address is frozen (EnforcementModule)
- The target address does not have enough active balance (ERC20EnforcementModule)
- If a `ruleEngine` is configured (ValidationModuleInternal)
- If the contract is in pause state
- If the contract is deactivated

	burn	batchBurn	burnFrom	burnAndMint	mint	batchMint	batchTransfer	crosschain burn	Crosschain mint	forcedTransfer
Module	ERC20Burn	ERC20Burn	CMTATBaseERC20CrossChain	CMTATBaseCommon	ERC20Mint	ERC20Mint	ERC20Mint	CMTATBaseERC20CrossChain	CMTATBaseERC20CrossChain	ERC20Enforcement
Module type	Core	Core	Options	Base module	Core	Core	Core	Options	Options	Extensions
Allow operation on a frozen address	☒	☒	☒	Same as burn & mint	☒	☒	☒	☒	☒	☑
Unfreeze missing funds if active balance is not enough (`ERC20EnforcementModule`)	☒	☒	☒	Same as burn & mint	-	-	☒	☒	-	☑
Call the `RuleEngine`	☑	☑	☑	Same as burn & mint	☑	☑	☑	☑	☑	☒
Authorised if contract is in pause state	☑	☑	☒	Same as burn & mint	☑	☑	☒	☒	☒	☑
Authorised if the contract is deactivated	☒	☒	☒	Same as burn & mint	☒	☒	☒	☒	☒	☑

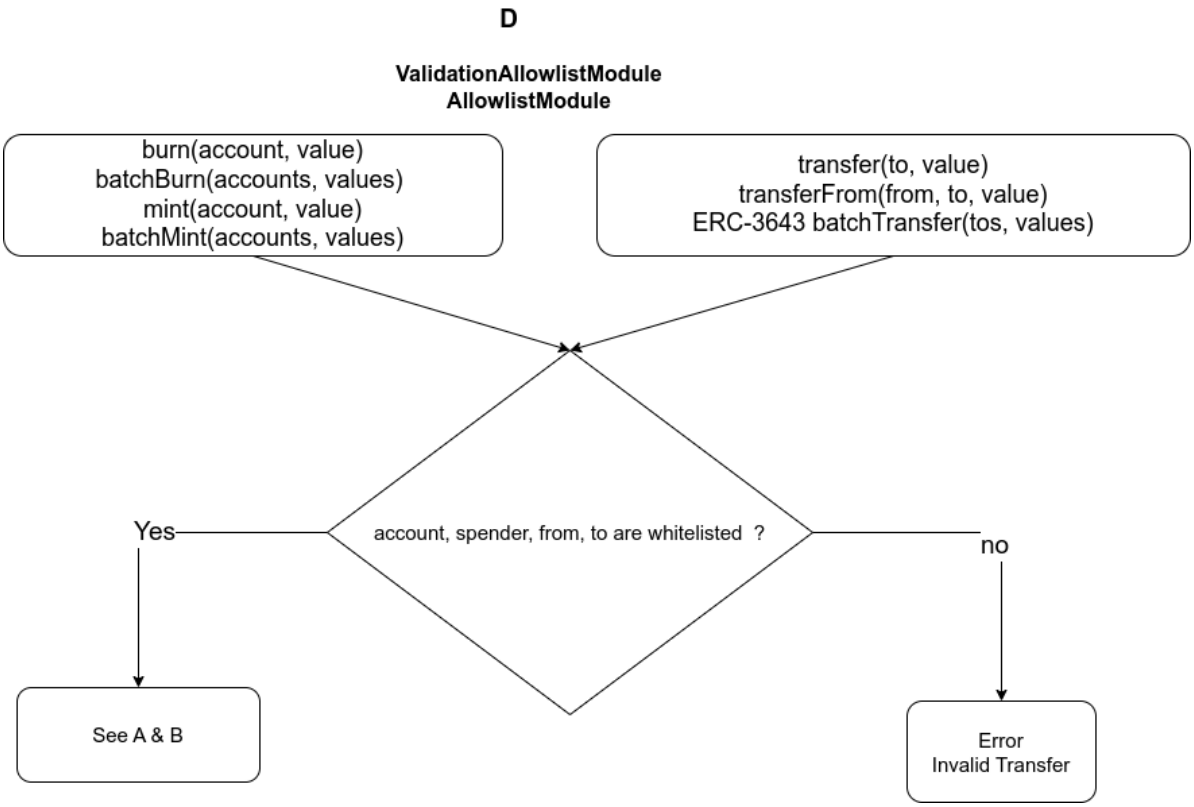
Note

Contrary to a `mint` operation, the function `batchTransfer` will perform the compliance check on the `from` address, which will be an address with the minter role. Another difference is the function will revert if the contract is in pause state.

Allowlist (whitelist) module

With the `Allowlist` module and the associated `ValidationModuleAllowlist`, a supplementary check will be performed on the concerned address to determine if they are in the allowlist.

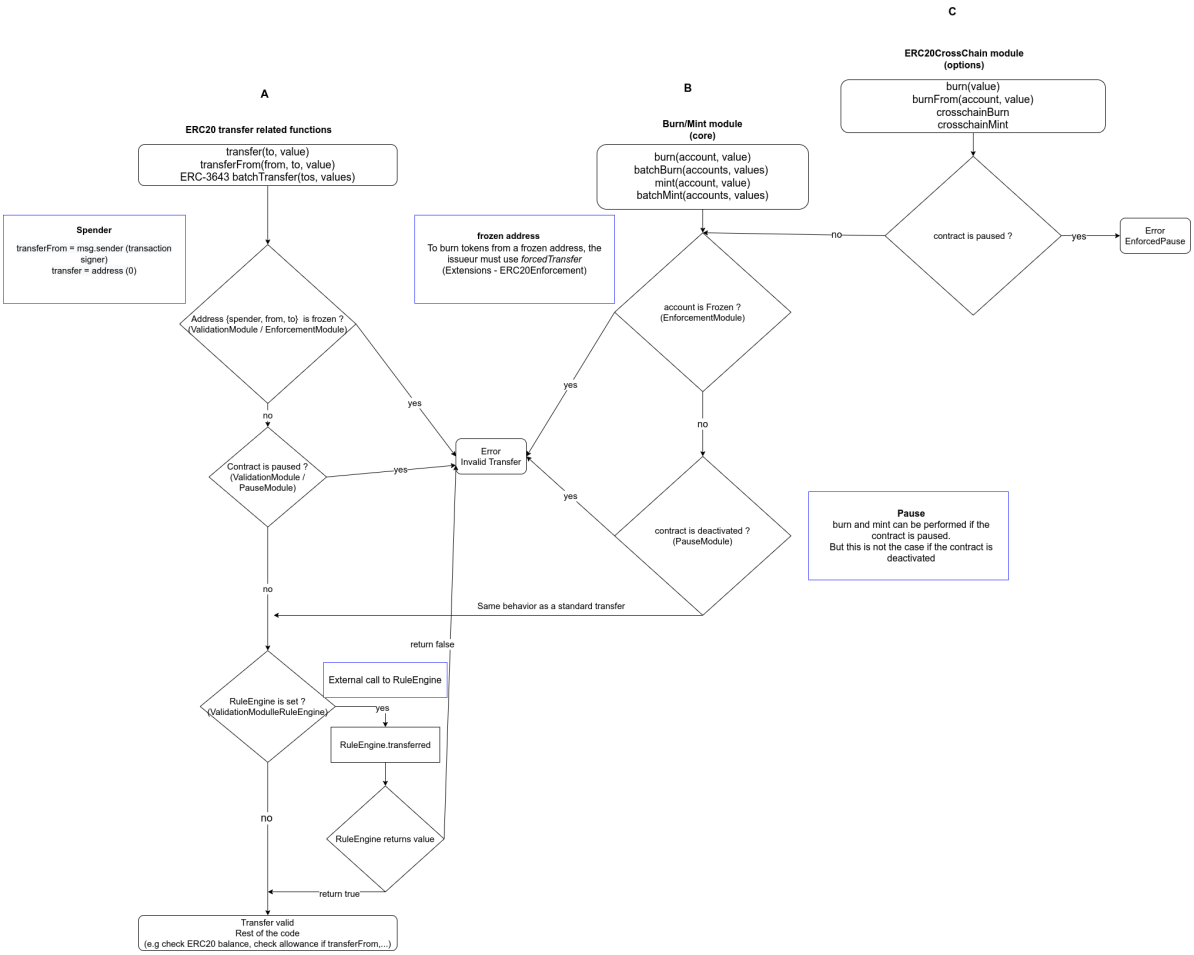
```
interface IAllowlistModule {
    /* ============ Events ============ */
    /**
     * @notice Emitted when an address is added to or removed from the
allowlist
     */
    event AddressAddedToAllowlist(address indexed account, bool indexed status,
address indexed enforcer, bytes data);
    /**
     * @notice Emitted when the allowlist is enabled or disabled
     */
    event AllowlistEnableStatus(address indexed operator, bool status);
    /* ============ Functions ============ */
    /**
     * @notice Checks if an account is allowlisted
     */
    function isAllowlisted(address account) external view returns (bool);
    /**
     * @notice Adds or removes an address from the allowlist
     */
    function setAddressAllowlist(address account, bool status) external;

    /**
     * @notice Adds or removes an address from the allowlist with additional
data
     */
    function setAddressAllowlist(address account, bool status, bytes calldata
data) external;
    /**
    * @notice Batch version of {setAddressAllowlist}
    */
    function batchSetAddressAllowlist(address[] calldata accounts, bool[]
calldata status) external;
    /**
     * @notice Enables or disables the allowlist
     */
    function enableAllowlist(bool status) external;

    /**
     * @notice Returns whether the allowlist is currently enabled
     */
    function isAllowlistEnabled() external view returns (bool);
}
```



Schema

Here a schema describing the different check performed during:

- `transfer`, `transferFrom` and `batchTransfer`
- `burn` / `mint` (supply management)
- `burn` / `mint` for crosschain transfers



Supply management

Event

Here the list of events emitted by functions, which modify the total supply.

Name	Defined	Standard	Concerned functions
`Transfer(address indexed from, address indexed to, uint256 value)`	IERC20 (OpenZeppelin)	ERC-20	All functions which impact the supply because a burn/mint is a transfer
`Mint(address indexed account, uint256 value, bytes data)`	IERC7551Mint	ERC-7551 (draft)	`mint` (ERC20MintModule)
`BatchMint( address indexed minter, address[] accounts, uint256[] values`		-	`batchMint` (ERC20MintModule)
`Burn(address indexed account, uint256 value, bytes data)`	IERC7551Burn	ERC-7551 (draft)	`burn` (ERC20BurnModule)
`BatchBurn(address indexed burner, address[] accounts, uint256[] values)`		-	`batchMint` (ERC20BurnModule)
`BurnFrom(address indexed burner, address indexed account, address indexed spender, uint256 value)`	IBurnERC20	-	`burnFrom(address account, uint256 value)` `burn(uint256 value)` (ERC20CrossChain)

Name	Defined	Standard	Concerned functions
`CrosschainMint(address indexed to, uint256 value, address indexed sender)`	IERC7551	ERC-7802	`crosschainMint` (ERC20CrossChain)
`CrosschainBurn(address indexed from, uint256 value, address indexed sender)`	IERC7551	ERC-7802	`crosschainMint` (ERC20CrossChain)
`Enforcement (address indexed enforcer, address indexed account, uint256 amount, bytes data)` (Enforcement)	IERC7551ERC20EnforcementEvent	ERC-7551 (draft)	`forcedTransfer` (ERC20EnforcementModule) `forcedBurn` (CMTATBaseCore)
`Spend(address indexed account, address indexed spender, uint256 value)`	IERC20Allowance	-	`transferFrom` (ERC20BaseModule) `transferFrom` don't change the supply `burnFrom(address account, uint256 value)`

Burn (ERC20BurnModule)

Core modue

ERC-3643

```
interface IERC3643Burn{
    /**
    *  @notice Burns tokens from a given address, by transferring them to address(0)
    */
    function burn(address account,uint256 value) external;
    /**
    * @notice Batch version of {burn}
    */
    function batchBurn(address[] calldata accounts,uint256[] calldata values) external;
}
```

ERC-7551

```
interface IERC7551Burn {
    /**
    * @notice Emitted when the specified `value` amount of tokens owned by
`owner`are destroyed with the given `data`
    */
    event Burn(address indexed burner, address indexed account, uint256 value,
bytes data);
    /**
    *  @notice Burns tokens from a given address, by transferring them to
address(0)
    */
    function burn(address account, uint256 amount, bytes calldata data)
external;
}
```

Mint (ERC20MintModule)

Core module

ERC-3643

```
interface IERC3643Mint{
    /**
    * @notice Creates a `value` amount of tokens and assigns them to `account`,
by transferring it from address(0)
    */
    function mint(address account, uint256 value) external;
    /**
    * @notice batch version of {mint}
    */
    function batchMint( address[] calldata accounts,uint256[] calldata values)
external;
}
```

ERC7551

```
interface IERC7551Mint {
    /**
     * @notice Emitted when the specified  `value` amount of new tokens are
created and
     * allocated to the specified `account`.
     */
    event Mint(address indexed minter, address indexed account, uint256 value,
bytes data);
    /**
    * @notice Creates a `value` amount of tokens and assigns them to `account`,
by transferring it from address(0)
    */
    function mint(address account, uint256 value, bytes calldata data)
external;
}
```

Cross-chain (ERC20Crosschain)

This part is implemented in the option module `ERC20CrossChain`

BurnFrom / burn

```
/**
 * @notice Standard interface for token burning operations with allowance.
 */
interface IBurnFromERC20 {
   /** ============ Events ============ **/
    /**
     * @notice Emitted when a spender burns tokens on behalf of an account,
reducing the spender's allowance.
    */
    event BurnFrom(address indexed burner, address indexed account, address
indexed spender, uint256 value);

    /** ============ Functions ============ **/
   /**
     * @notice Burns a specified amount of tokens from a given account,
deducting from the caller's allowance.
     */
  function burnFrom(address account, uint256 value) external;

    /**
    * @notice Burns a specified amount of tokens from the caller's own balance.
    * @param value The number of tokens to burn.
    * @dev This function is restricted to authorized roles.
    */
    function burn(uint256 value) external;
}
```

ERC-7802

See the dedicated section (at the beginning of this document)

Access control

in the different `CMTATBase` modules, the function responsible to manage the access control are overridden to forbid `self burn`.

It means that a token holder can not burn its own tokens.

Example (ERC20CrossChainModule):

```
function _authorizeBurnFrom() internal virtual override(ERC20CrossChainModule)
onlyRole(BURNER_FROM_ROLE) whenNotPaused{}
```

Reason

It's deliberate that only the issuer (and not the tokenholder) can cancel a token, and that this corresponds to a legal requirement in several countries.

Indeed, once issued, a security can only be cancelled by its issuer, not by its holder. Since the token serves as a vehicle for the security, the same must apply to the token itself. An investor wishing to "get rid of" a token must transfer it to the issuer, who can then cancel it when the law allows.

Alternative

You can still allow `self burn` by creating a new function or by overriden the corresponding functions.

Manage on-chain document

Terms

Tokenization terms are defined by the extension module `ExtraInformationModule`

The term is made of:

- A name (string)
- An `IERC1643.Document` document, which means:
  - A string uri (optional)
  - The document hash (optional)
  - The last on-chain modification date (set by the smart contract)

```
interface IERC1643 {
    struct Document {
        string uri;
        bytes32 documentHash;
        uint256 lastModified;
    }
   // rest of the interface
}
interface ICMTATBase {
     /*
     * @dev A reference to (e.g. in the form of an Internet address) or a hash
of the tokenization terms
     */
     struct Terms {
        string name;
        IERC1643.Document doc;
    }
    event Term(Terms newTerm);
    /*
    * @notice returns tokenization terms
    */
    function terms() external view returns (Terms memory);
    /*
    * @notice set tokenization terms
    */
    function setTerms(IERC1643CMTAT.DocumentInfo calldata terms_) external;
}
```

Additional documents through ERC1643 and DocumentEngine

Additional documents can be added through the `DocumentEngine`

For more information, see the section dedicated to the `DocumentEngine`

Cross-chain transfers (ERC-7802, CCIP-CCT)

Chainlink CCIP - CCT

CMTAT implements the required function of the Cross-Chain Token Standard (CCT) which means:

- CMTAT CCIP admin can enable the token in CCIP, without the need of requesting assistance to Chainlink.
- Chainlink CCIP pool can perform the relevant mint and burn operations on CMTAT tokens.

See also docs.chain.link - Cross Chain Token (EVM)

Registration functions

CMTAT implements the following function `getCCIPAdmin()` to return the address authorized to register the token in CCIP.

The alternative function proposed by CCIP, `owner`, is not implemented by CMTAT but this could be done easily. Note that `getCCIPAdmin` is the recommended function to use in the CCIP documentation.

Transfer functions

Here is the list of functions required to implement CCT and be compatible with CCIP.

		Implemented	CCIP Pool BurnMint Requirements	CCIP Pool Lock-Release requirements	Pausable	CMTAT Module	Role
Register CCIP token							
	`owner()`	☒	-	-		-	
	`getCCIPAdmin()`	☑	-	-		CCIPModule	-
Burn and Mint Requirements							
	`mint(address account, uint256 amount)`	☑	☑	☒	☒	MintModule (Core module)	MINTER_ROLE
	`burn(uint256 amount)`	☑	☑	☒	☒	ERC20CrossChain	BURNER_FROM_ROLE
	ERC-20 `decimals()`	☑	☑	☑	-	ERC20BaseModule (Core module)	
	ERC-20 `balanceOf(address account)`	☑	☑	☑	-	OpenZeppelin inheritance	
	`burnFrom(address account, uint256 amount)`	☑	☑	☒	☑	ERC20CrossChain	BURNER_FROM_ROLE

Note:

- `Lock and Mint` and `Burn and Unlock` models are also compatible with CMTAT through the implementation of `Burn and Mint` requirements.
- `Lock and Unlock` model does not need specific requirement from the token contract.
- The admin must grant the required permissions to mint/burn to the CCIP token pool.

- Pausing the contract through the PauseModule will not affect the mint and burn functions of the MintModule and BurnModule. The alternative solution in this case is to revoke the MINTER_ROLE and BURNER_ROLE from the relevant addresses to prevent minting and burning.

CMTAT implementation

Here is the list of implemented functions and their respective modules.

```
// ERC20BaseModule
function decimals() public view virtual override(ERC20Upgradeable) returns
(uint8)
// OpenZeppelin ERC20Upgrdeable
function balanceOf(address account) public view virtual returns (uint256)
// CCIPModule
function setCCIPAdmin(address newAdmin) public virtual onlyCCIPSetAdmin
function getCCIPAdmin() public view virtual returns (address)
// MintModule
function mint(address account, uint256 value) public virtual
override(IERC5679Mint) onlyMinter
// ERC20CrossChain
function burnFrom(address account, uint256 value) public virtual
override(IBurnFromERC20) onlyBurnerFrom
function burn(uint256 value) public virtual onlyBurnerFrom
```

Example

CMTAT-CCIP repository, made by Nox Labs, contains a collection of Foundry scripts designed to simplify and show deployment of a CMTAT token (v3.1.0) with CCIP contracts.

Optimism superchain ERC-20 (ERC-7802)

> CMTAT implements ERC-7802 in the option module `ERC20CrossChain`

The `SuperchainTokenBridge` uses ERC-7802 to enable asset interoperability within the Superchain.

- Asset interoperability allows tokens to move across the Superchain by burning tokens on the source chain and minting an equivalent amount on the destination chain.
- Instead of wrapping assets, this mechanism effectively "teleports" tokens between chains in the Superchain.

Reference: docs.optimism.io/interop/superchain-erc20

Initiating message (source chain)

> Example of use

1. The user (or a contract) calls `SuperchainTokenBridge.sendERC20`.
2. The token bridge calls the function `CMTAT.crosschainBurn` to burn those tokens on the source chain.
3. The source token bridge calls `SuperchainTokenBridge.relayERC20` on the destination token bridge. This call is relayed using `L2ToL2CrossDomainMessenger`. The call is *initiated* here, by emitting an initiating message. It will be executed later, after the destination chain receives an executing message to `L2ToL2CrossDomainMessenger`.

Executing message (destination chain)

1. The autorelayer (or the user, or any offchain entity) sends an executing message to `L2ToL2CrossDomainMessenger` to relay the message.
2. The destination token bridge calls `CMTAT.crosschainMint` to mint tokens for the user/contract that called `SuperchainTokenBridge.sendERC20` originally.

Requirement

- You must allow the `SuperchainTokenBridge` to call `crosschainMint` and `crosschainBurn`. No additional permission or role setup is required.
- Deploy the `CMTAT` at the same address on every chain in the Superchain where you want your token to be available. If you do not deploy the contract to a specific destination chain, users will be unable to successfully move their tokens to that chain.

Deployment model

Contracts for deployment are available in the directory contracts/deployment

Summary tab

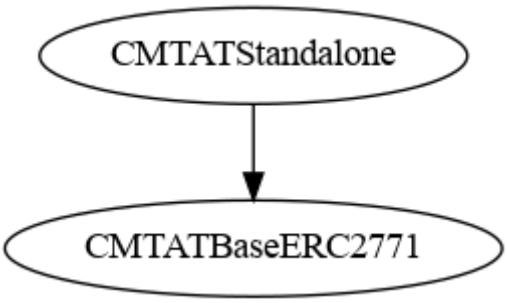
CMTAT Model	Description	Standalone/Proxy	Contract	Note
CMTAT Standard	Deployment without proxy (immutable)	Standalone	CMTATStandalone	Core & extension module without Debt, Allowlist, ERC-3643 and UUPS Include also the option module `ERC2771`, as well as `ERC20CrossChain` support
	Deployment with a standard proxy (Transparent or Beacon Proxy)	Upgradeable	CMTATUpgradeable	-
Upgradeable UUPS	Deployment with a UUPS proxy	Only upgradeable	CMTATUpgradeableUUPS	Same as standard version, but adds also the UUPS proxy support
ERC-1363	Implements ERC-1363	Standalone	CMTATStandaloneERC1363	Same as standard version, but adds also the support of `ERC-1363`
	-	Upgradeable	CMTATUpgradeableERC1363	
Light	Only core modules	Standalone	CMTATStandaloneLight	The core features (i.e., minting, burning,address freeze / blacklisting, pause) without additional functions required by equities and debt instruments (e.g., document management, snapshot, partial freeze of balances).
		Upgradeable	CMTATUpgradeableLight	

CMTAT Model	Description	Standalone/Proxy	Contract	Note
Debt	Set Debt information and Credit Events	Standalone	CMTATStandaloneDebt	Add the debt support. Contrary to the standard version, it does not include the module `ERC2771Module` and the support of `ERC20CrossChain`
		Upgradeable	CMTATUpgradeableDebt	-
Allowlist	Restrict transfer to an allowlist (whitelist)	Standalone	CMTATStandaloneAllowlist	Contrary to the standard version, it does not include the `RuleEng`ERC-1404` support (ValidationModuleERC1404) & ERC20Crosschain
		Upgradeable	CMTATUpgradeableAllowlist	-
ERC7551	Deployment specific for ERC-7551	Standalone	CMTATStandaloneERC7551	Add support of `ERC7551Module`
		Upgradeable	CMTATUpgradeableERC7551	-
CMTAT with snapshots	Deployment version that performs time-based snapshots directly on-chain and without relying on the external contract `SnapshotEngine`	Upgradeable	CMTA - SnapshotEngine (external repository)	

Standard Standalone

To deploy CMTAT without a proxy, in standalone mode, you need to use the contract version `CMTATStandalone`.

Here is the surya inheritance schema:



Upgradeable (with a proxy)

The CMTAT supports deployment via a proxy contract. Furthermore, using a proxy permits to upgrade the contract, using a standard proxy upgrade pattern.

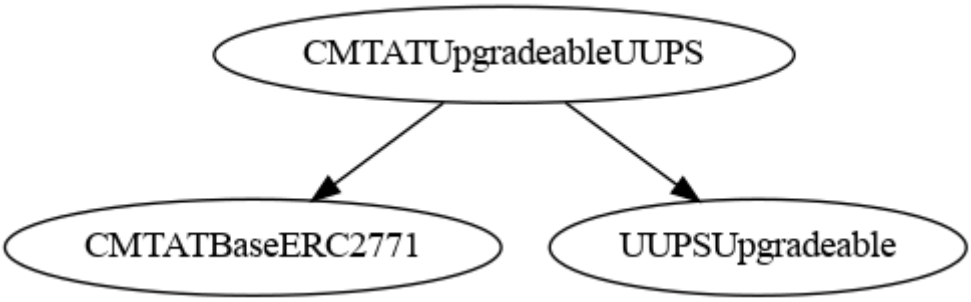
- The implementation contract to use with a TransparentProxy is the `CMTATUpgradeable`.
- The implementation contract to use with a UUPSProxy is the `CMTATUpgradeableUUPS`.

Please see the OpenZeppelin upgradeable contracts documentation for more information about the proxy requirements applied to the contract.

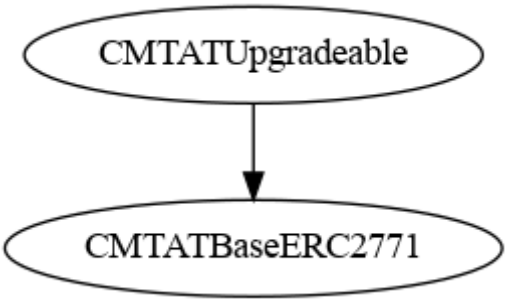
See the OpenZeppelin Upgrades plugins for more information about plugin upgrades in general.

Inheritance

- UUPS



- Proxy standard



Implementation details

Storage

CMTAT also implements the standard ERC-7201 to manage the storage location. See this article by RareSkills for more information.

Modules	Variable	bytes32 Value
Internal		
AllowlistModuleInternal	AllowlistModuleInternalStorageLocation	0x53076eaf2d1e2f915f2e0487c9f92cca686c37fd47bf11f95f0da313b2809800
EnforcementModuleInternal	EnforcementModuleInternalStorageLocatio	0x0c7bc8a17be064111d299d7669f49519cb26c58611b72d9f6ccc40a1e1184e00
ERC20EnforcementModuleInternal	ERC20EnforcementModuleStorageLocation	0x9d8059a24cb596f1948a937c2c163cf14465c2a24abfd3cd009eec4ac4c39800
	ValidationModuleRuleEngineStorageLocation	0x77c8cc897d160e7bf5b10921804e357da17ae27460d4a6b5d9b27ffddf159d00
Core		
ERC20BaseModule	ERC20BaseModuleStorageLocation	0x9bd8d607565c0370ae5f91651ca67fd26d4438022bf72037316600e29e6a3a00
PauseModule		0xab1527b6135145d8da1edcbd6b7b270624e17f2b41c74a8c746ff388ad454700
Extension		
DocumentEngineModule	DocumentEngineModuleStorageLocation	0xbd0905600c85d707dc53eba2e146c1c2527cd32ac3ff6b86846155151b3e2700
ExtraInformationModule	ExtraInformationModuleStorageLocation	0xd2d5d34c4a4dea00599692d3257c0aebc5e0359176118cd2364ab9b008c2d100
SnapshotEngineModule	SnapshotEngineModuleStorageLocation	0x1387b97dfab601d3023cb57858a6be29329babb05c85597ddbe4926c1193a900
Options		
CCIPModule	CCIPModuleStorageLocation	0x364fbfd89c0eee55bbc8dd10b1a9bf3e04fba9f3ee606f4c79a82f9941ad7a00
DebtModule	DebtModuleStorageLocation	0xf8a315cc5f2213f6481729acd86e55db7ccc930120ccf9fb78b53dcce75f7c00
ERC7551Module	ERC7551ModuleStorageLocation	0x2727314c926b592b6f70e7d6d2e4677ebcac070f293306927f71fe77858eec00

Initialize functions

Inside the public initializer function to initialize your proxy, you have to call the different functions `__{ContractName}_init_unchained`.

Do not forget to call the functions `init_unchained` from the parent initializer if you create your own contract from the different modules.

For wrapper modules, we have removed the public function `{ContractName}_init` when they are not useful to reduce the size of the contracts.

As indicated in the OpenZeppelin documentation:

> Initializer functions are not linearized by the compiler like constructors. Because of this, each `__{ContractName}_init` function embeds the linearized calls to all parent initializers. As a consequence, calling two of these `init` functions can potentially initialize the same contract twice.
>
> The function `__{ContractName}_init_unchained` found in every contract is the initializer function minus the calls to parent initializers, and can be used to avoid the double initialization problem, but doing this manually is not recommended. We hope to be able to implement safety checks for this in future versions of the Upgrades Plugins.

ERC-1363

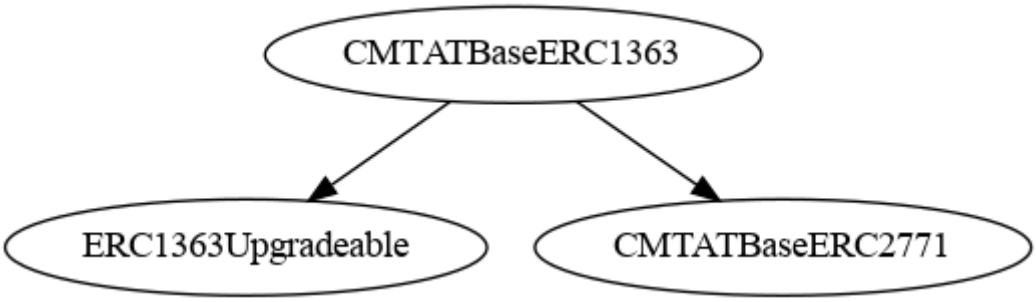
ERC-1363 is an extension interface for ERC-20 tokens that supports executing code on a recipient contract after transfers, or code on a spender contract after approvals, in a single transaction.

Two dedicated versions (proxy and standalone) implementing this standard are available.

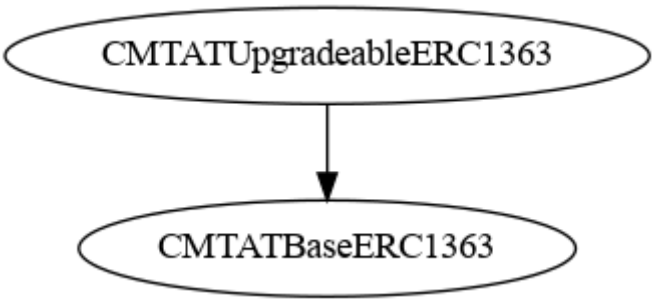
More information on this standard here: erc1363.org, RareSkills - ERC-1363

Inheritance

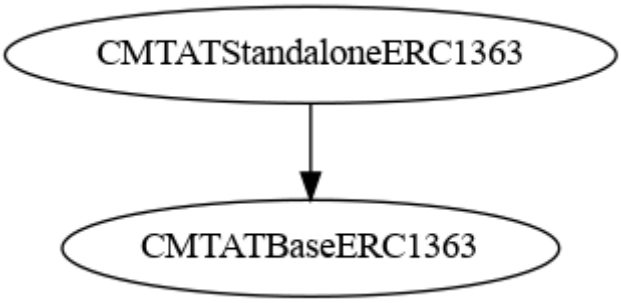
- CMTAT ERC-1363 Base



- CMTAT Upgradeable ERC-1363



- CMTAT Standalone ERC-1363



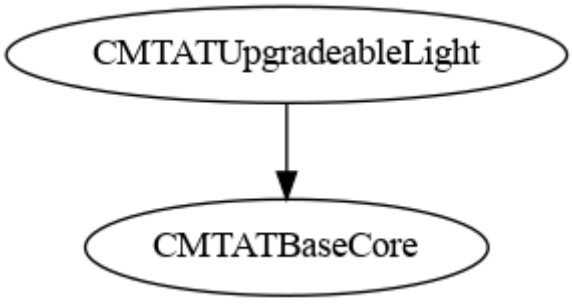
Light version

The light version only includes core modules.

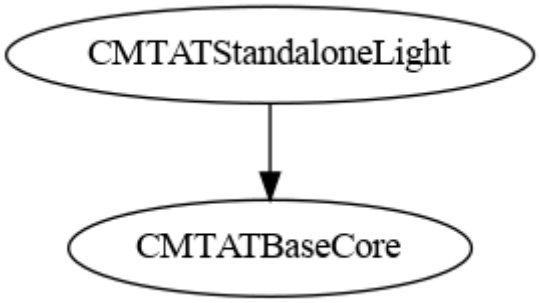
It also includes a function `forceBurn` to allow the admin to burn a token from a frozen address. This function is not required for deployment versions which include the extension module `ERC20EnforcementModule` because this module contains a function `forcedTransfer` which can be used instead.

If the address is not frozen, it is also possible to perform a burn-and-mint atomically through the function `burnAndMint` like the deployment standard versions

- CMTAT Upgradeable Light



- CMTAT Standalone Light



- CMTATBaseCore

Debt version

This deployment version includes the optional module `DebtModule` and `DebtEngineModule` which allows for the first to store information related to the debt instrument and credit events inside the smart contract, or through an external contract called `DebtEngine` for the second.

See CMTAT - Standard for the tokenization of debt instruments using distributed ledger technology

Struct

The debt information are defined by the struct `ICMTATDebt` in ICMTAT.sol

```
interface ICMTATDebt {
    struct DebtInformation {
        DebtIdentifier debtIdentifier;
        DebtInstrument debtInstrument;
    }
    struct DebtIdentifier {
        string issuerName;
        string issuerDescription;
        string guarantor;
        string debtHolder;
    }
    struct DebtInstrument {
        // uint256
        uint256 interestRate;
        uint256 parValue;
        uint256 minimumDenomination;
        // string
        string issuanceDate;
        string maturityDate;
        string couponPaymentFrequency;
        string interestScheduleFormat;
        string interestPaymentDate;
        string dayCountConvention;
        string businessDayConvention;
        string currency;
        // address
        address currencyContract;
    }
    function debt() external view returns(DebtInformation memory);
}
```

Debt Identifier

Information on the issuer and other persons involved.

Defined by the struct `DebtIdentifier` in ICMTAT.sol

Field name	Type	Description
issuerName	string	Issuer identifier (legal entity identifier [LEI] or, if unavailable, Swiss entity identification number [UID] or equivalent)
issuerDescription	string	-
guarantor	string	Guarantor identifier (legal entity identifier [LEI] or, if unavailable, Swiss entity identification number [UID] or equivalent), if applicable
debtHolder	string	Debtholders representative identifier (legal entity identifier [LEI] or, if unavailable, Swiss entity identification number [UID] or equivalent), if applicable

Debt Instrument

Information on the Instruments.

Defined by the struct `DebtInstrument` in ICMTAT.sol

Field name	Type	Description
interestRate	uint256	-
parValue	uint256	-
minimumDenomination	uint256	-
issuanceDate	string	-
maturityDate	string	-
couponPaymentFrequency	string	-
interestScheduleFormat	string	The purpose of the interest schedule is to set, within 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/....
interestPaymentDate	string	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

Field name	Type	Description
dayCountConvention	string	-
businessDayConvention	string	-
currency	string	-
currencyContract	address	-

Credit Events

Defined by the struct `CreditEvents` in ICMTAT.sol.

Similar to the debt information, Credit Events can be set directly inside the smart contract (`DebtModule`) or through the external contract `DebtEngine` (`DebtEngineModule`).

```
interface ICMTATCreditEvents {
    function creditEvents() external view returns(CreditEvents memory);
    struct CreditEvents {
        bool flagDefault;
        bool flagRedeemed;
        string rating;
    }
}
```

	Type
flagDefault	bool
flagRedeemed	bool
rating	string

Specification

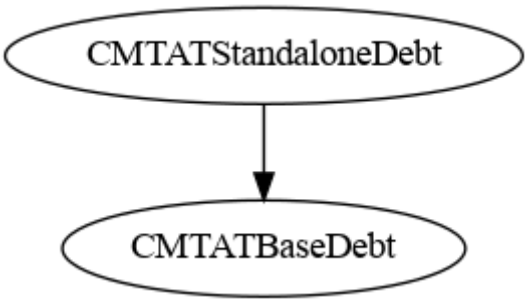
Here are the different fields and functions to read and store the related debt information and Credit Events.

	Module	Function	Type [Read/Write]	Override by DebtEngine	Internal field
Debt Identifier					
	DebtModule DebtEngineModule	debt()	Read	☑	`_debt`
	DebtModule	setDebt(...)	Write	☒	`_debt`
Debt Instrument					
	DebtModule DebtEngineModule	debt()	Read	☑	`_debt`
	DebtModule	setDebt(...) setDebtInstrument(...)	Write	☒	`_debt`

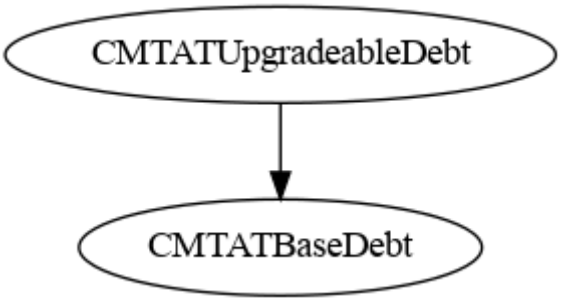
	Module	Function	Type [Read/Write]	Override by DebtEngine	Internal field
Credit Events	DebtEngineModule	creditEvents()	Read	☑	`_creditEvents`
	DebtModule	setCreditEvents(...)	Write	☒	`_creditEvents`

Schema

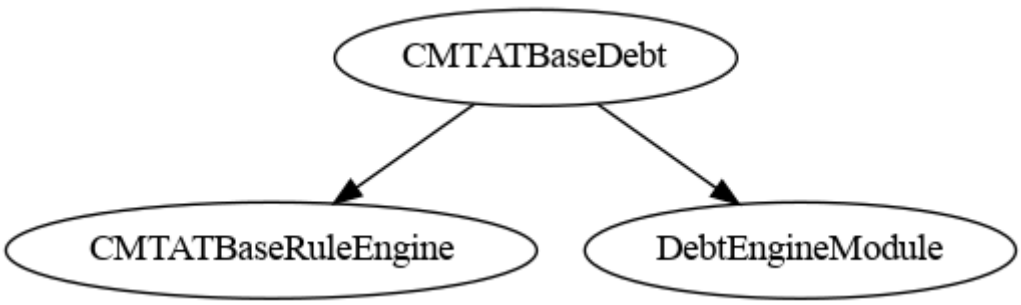
- CMTAT Standalone Debt



- CMTAT Upgradeable Debt



- CMTAT Base Debt



Allowlist

The Allowlist deployment version allows to restrict transfer to token holders present inside an allowlist (whitelist) maintained inside the smart contract.

For this purpose, a specific Validation controller is used called `ValidationModuleAllowlist` as well as a specific option module `AllowlistModule`.

As a result, with this deployment version, it is not possible to set a `RuleEngine` and the contract does not implement the standard `ERC-1404`.

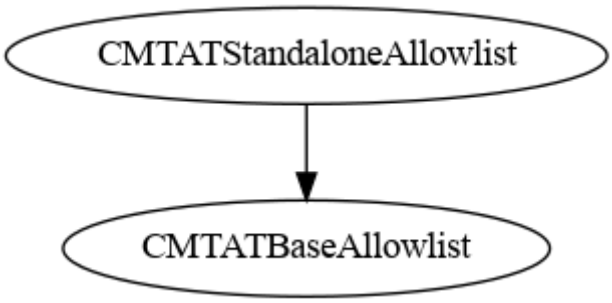
More information regarding the Ethereum API available in the [Allowlist module documentation]

How to use it ?

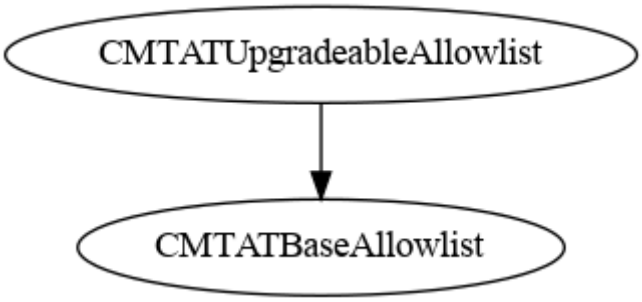
1. Select the deployment version you want: `CMTATStandaloneAllowlist` or `CMTATUpgradeableAllowlist`
2. Once the contract is deployed, with an authorized user (default admin or an address with the `ALLOWLIST_ROLE`) enables the `allowlist` by calling the function `enableAllowlist` with true as status.
    - Once this is done, all transfers (including `mint` and `burn`) will be rejected if the origin or target address is not in the `allowlist`
        - For a mint operation, the contract authorized the origin address zero by default.
        - For a burn operation, the operation will be rejected if the target account is not in the `allowlist`. In this case, the issuer must use the function `forcedTransfer` to burn the tokens.
    - It is possible to disable the use of the allowlist by calling the same function `enableAllowlist` with false as status.
3. Add the different addresses in the `allowlist` by calling the functions `setAddressAllowlist` and `batchSetAddressAllowlist`. It is possible to call theses functions even if the `allowlist` is not enabled.

Inheritance

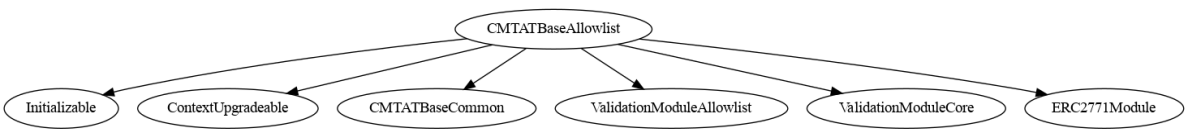
- CMTAT Standalone Allowlist



- CMTAT Upgradeable Allowlist



- CMAT base Allowlist



Factory

Factory contracts are available to deploy the CMTAT with a beacon proxy, a transparent proxy or an UUPS proxy.

These contracts have now their own GitHub project: CMTAT Factory

CMTAT version	CMTAT Factory
CMTAT v3.0.0	CMTAT Factory v0.2.0 (unaudited)
CMTAT v2.5.0 / v2.5.1 (unaudited)	Available within CMTAT see contracts/deployment (unaudited)
CMTAT 2.3.0 (audited)	Not available
CMTAT 1.0 (audited)	Not available

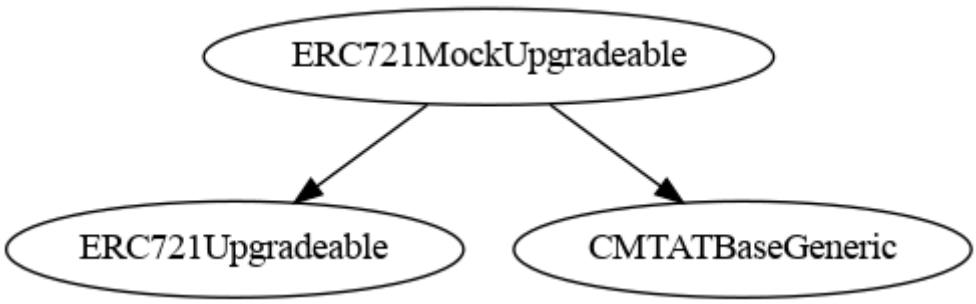
Further reading: Taurus - Making CMTAT Tokenization More Scalable and Cost-Effective with Proxy and Factory Contracts (version used CMTAT v2.5.1)

Deployment for other types of tokens (ERC-721, ERC-1155, ...)

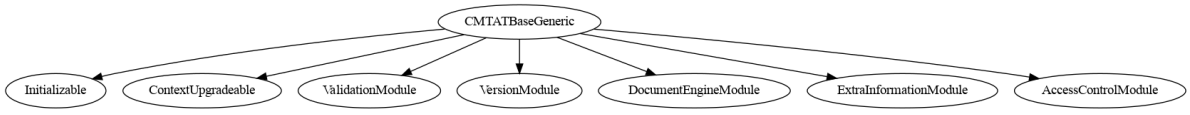
Deployment version using another type of token than ERC-20 (e.g ERC-721) or with a different logic (e.g ZamaFHE - EncryptedERC20) can be built by using the base contract `CMTATBaseGeneric`. This base contract inherits from several non-ERC-20 modules

Currently, there is no available version but a mock contract which implements ERC-721 with `CMTATBaseGeneric` is available in the mock directory: EC721MockUpgradeable.sol

- ERC721MockUpgradeable



- CMTATBaseGeneric



Documentation

The documentation is available in the directory `doc`

Here a summary of the main documents

Document	Files
Documentation of the modules API.	modules
How to use the project + toolchains	USAGE.md
FAQ	FAQ.md
Crosschain transfers	crosschain-bridge-support.md

CMTA provides further documentation describing the CMTAT framework in a platform-agnostic way, and covering legal aspects, see

- CMTA Token (CMTAT)
- Standard for the tokenization of shares of Swiss corporations using the distributed ledger technology
- Standard for the tokenization of debt instruments using distributed ledger technology

Further reading

- Solidity (EVM version)
  - CMTA - A comparison of different security token standards
  - Taurus - Security Token Standards: A Closer Look at CMTAT
  - Taurus - Equity Tokenization: How to Pay Dividend On-Chain Using CMTAT (CMTAT v2.4.0)
  - Taurus - Token Transfer Management: How to Apply Restrictions with CMTAT and ERC-1404 (CMTAT v2.4.0)
  - Taurus - Making CMTAT Tokenization More Scalable and Cost-Effective with Proxy and Factory Contracts (CMTAT v2.5.1)
  - Taurus - Conditional Transfers with CMTAT & Taurus-CAPITAL: A Step-by-Step Guide (CMTAT v2.5.0)
- Aztec
  - Taurus - Addressing the Privacy and Compliance Challenge in Public Blockchain Token Transactions (Aztec)
  - Taurus Deploys the First Private Stablecoin Contract

Security

Vulnerability disclosure

Please see SECURITY.md.

Module

Access control is managed thanks to the module `AccessControlModule`.

See AccessControlModule.sol

Audit

CMTAT is regularly audited by globally recognised firm specialised in smart contracts security.

Only the version audited should be used in production.

Version	Security Audit Firm
CMTAT v3.0.0	Halborn
CMTAT v2.3.0	ABDKConsulting
CMTAT v1.0.0	ABDKConsulting

Out of scope

Mocks contracts in the directory contracts/mocks are not audited and are not intended for use in production.

They are only used for testing.

First audit - September 2021 [ABDK]

Fixed version: 1.0

Fixes of security issues discovered by the initial audit were reviewed by ABDK and confirmed to be effective, as certified by the report released on September 10, 2021, covering version c3afd7b of the contracts.
Version 1.0 includes additional fixes of minor issues, compared to the version retested.

A summary of all fixes and decisions taken is available in the file CMTAT-Audit-20210910-summary.pdf

Second audit - March 2023 [ABDK]

Fixed version: v2.3.0

The second audit covered version 2.2.

Version v2.3.0 contains the different fixes and improvements related to this audit.

The report is available in ABDK_CMTA_CMTATRuleEngine_v_1_0.pdf.

Third audit - July 2025 [Halborn]

This audit has been made by Halborn.

Fixed version: v3.0.0

The third audit covered version v3.0.0-rc5.

Version v3.0.0 contains the different fixes and improvements related to this audit.

The report is available in
Taurus_CMTAT_Smart_Contract_Security_Assessment_Report_Halborn.pdf.

> After the 1st audit phase, we made another fix to perform compliance check with all batch functions. See commits - 198d0194a0eef526b0a33cb625f6227da07608d4. This fix was also reviewed by Halborn.

Tools

> More details are available in the file USAGE.md

Aderyn

Here are the reports produced by Aderyn:

Version	File
v3.1.0	v3.1.0-aderyn-report.md
v3.0.0	v3.0.0-aderyn-report.md

Slither

Here are the reports produced by Slither:

Version	File
v3.1.0	v3.1.0-slither-report.md
v3.0.0	v3.0.0-slither-report.md
v2.5.0	v2.5.0-slither-report.md
v2.3.0	v2.3.0-slither-report.md

Mythril

Here are the reports produced by Mythril

Version	File
v3.0.0	Mythril currently generates a fatal error, impossible to run the tool
v2.5.0	mythril-report-standalone.md mythril-report-proxy.md

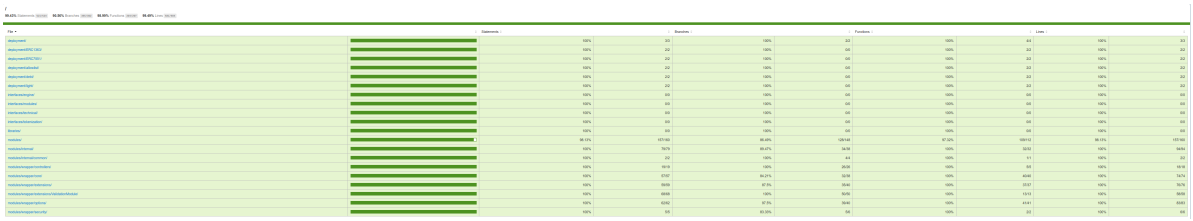
Nethermind Audit Agent

Here are the reports produced by Nethermind Audit Agent

Version	File
v3.1.0	nethermind-audit-agent/v3.1.0
v3.0.0-rc5	nethermind-audit-agent/v3.0.0-rc5

Test

A code coverage is available in index.html.



Usage

More details are available in the file USAGE.md

Solidity style guideline

CMTAT follows the solidity style guideline present here: docs.soliditylang.org/en/latest/style-guide.html

- Orders of Functions

Functions are grouped according to their visibility and ordered:

```
1. constructor

2. receive function (if exists)

3. fallback function (if exists)

4. external

5. public

6. internal

7. private
```

Within a grouping, place the `view` and `pure` functions last

- Function declaration

```
1. Visibility
2. Mutability
3. Virtual
4. Override
5. Custom modifiers
```

Inside each contract, library or interface, use the following order:

```
1. Type declarations
2. State variables
3. Events
4. Errors
5. Modifiers
6. Functions
```

Configuration & toolchain

Details

The project is built with Hardhat and uses OpenZeppelin

- hardhat.config.js
  - Solidity v0.8.30
  - EVM version: Prague (Pectra upgrade)
  - Optimizer: true, 200 runs
- Package.json
  - OpenZeppelin Contracts (Node.js module): v5.4.0
  - OpenZeppelin Contracts Upgradeable (Node.js module): v5.4.0

Installation & Compilation

- Clone the repository

Clone the git repository, with the option `--recurse-submodules` to fetch the submodules:

```
git clone git@github.com:CMTA/CMTAT.git  --recurse-submodules
```

- Install node modules

```
npm install
```

- Run test

```
npx hardhat test
```

Hardhat

> Since the sunset of Truffle by Consensys, Hardhat is our main development environment.

To use Hardhat, the recommended way is to use the version installed as part of the node modules, via the `npx` command:

```
npx hardhat
```

Alternatively, you can install Hardhat globally:

```
npm install -g hardhat
```

Contract size

```
npm run-script size
```

```
> npx hardhat size-contracts

Nothing to compile
·------------------------------|------------------------------|------------------------------·
|  Solc version: 0.8.30        ·  Optimizer enabled: true     ·  Runs: 200                   |
·------------------------------|------------------------------|------------------------------·
|  Contract Name               ·  Deployed size (KiB) (change)  ·  Initcode size (KiB) (change)  |
·------------------------------|------------------------------|------------------------------·
|  Address                     ·                0.083 (0.000)  ·                0.132 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  Arrays                      ·                0.083 (0.000)  ·                0.132 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  Bytes                       ·                0.083 (0.000)  ·                0.132 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandalone             ·               21.089 (0.000)  ·               24.767 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneAllowlist    ·               18.448 (0.000)  ·               21.948 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneDebt         ·               23.926 (0.000)  ·               27.330 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneERC1363      ·               22.633 (0.000)  ·               26.339 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneERC7551      ·               21.689 (0.000)  ·               25.367 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneLight        ·               10.773 (0.000)  ·               12.510 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeable            ·               21.089 (0.000)  ·               21.415 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableAllowlist   ·               18.448 (0.000)  ·               18.774 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableDebt        ·               23.926 (0.000)  ·               24.135 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableERC1363     ·               22.633 (0.000)  ·               22.959 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableERC7551     ·               21.689 (0.000)  ·               22.016 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableLight       ·               10.773 (0.000)  ·               10.982 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableUUPS        ·               23.276 (0.000)  ·               23.629 (0.000)  |
·------------------------------|------------------------------|------------------------------·
```

Other implementations

Aztec (Noir)

A specific version is available for Aztec: Aztec Private CMTAT made by Taurus in collaboration with CMTA

- This version is not official in the sense that it was not approved formally by the CMTA
- See also Taurus - Addressing the Privacy and Compliance Challenge in Public Blockchain Token Transactions

Solana

Specification to deploy CMTAT compliant token on Solana are available in the repository CMTAT-Solana made by Taurus as an internal CMTA project in collaboration with Solana Foundation.

Starknet (Cairo)

A version for Starknet written in Cairo is currently under development by Sereel in collaboration with CMTA: 0xsereel/cairo-cmtat

Tezos

Two versions are available for the blockchain Tezos

- CMTAT FA2: Official version written in SmartPy made by AirGap in collaboration with CMTA.
- @ligo/cmtat: Unofficial version written in Ligo made by Frank Hillard.
    - See also Tokenization of securities on Tezos by Frank Hillard

Summary tab

If you create a version for another blockchains, feel free to use this summary tab to build a correspondence table between CMTAT framework, CMTAT Solidity version and your implementation.

CMTAT framework

In the below table, the CMTAT framework required features are mapped to Solidity features.

CMTAT framework mandatory functionalities	**CMTAT Solidity corresponding features**
Know total supply	ERC20 `totalSupply`
Know balance	ERC20 `balanceOf`
Transfer tokens	ERC20 `transfer`
Create tokens (mint)	`Mint/batchMint`
Cancel tokens (force burn)	`burn/batchBurn` *(Nb. we recommend to have a dedicated function to burn tokens without the token holder consent or from a frozen address*)
Pause tokens	Pause (*Nb. With CMTAT Solidity it is still possible to burn and mint while transfers are paused.)*
Unpause tokens	`unpause`
Deactivate contract	`deactivateContract`
Freeze	`setAddressFrozen` (previously `freeze`)
Unfreeze	`setAddressFrozen` (previously `unfreeze`)
Name attribute	ERC20 `name` attribute
Ticker symbol attribute	ERC20 `symbol` attribute
Token ID attribute	`tokenId`
Reference to legally required documentation	`terms` (document name, hash and uri with at least the uri)

Freeze

To be compatible with ERC-3643, the freeze functionality is implemented with only one function: `setAddressFrozen` which takes the target address and the frozen status (true/false).

However, for non-EVM blockchains, it could make clearer and makes more sense to separate the freeze and unfreeze (or `thaw`) functionality with two separates and distinct functions, such as:

```
freeze(address targetAddress)
unfreeze(address targetAddress)
```

CMTAT extended

In the below table, the CMTAT framework extendedfeatures are mapped to Solidity features.

CMTAT Functionalities	CMTAT Solidity corresponding features	CMTAT Allowlist	CMTAT Light	CMTAT Debt	CMTAT Standard
On-chain snapshot	`snapshotModule` & `snapshotEngine`	☑	☒	☑	☑
Forced Transfer	`forcedTransfer`	☑	☒	☑	☑
Forced burn	`forcedBurn`	☒	☑	☒	☒
Freeze partial token	`freezePartialTokens` / `unfreezePartialTokens`	☑	☒	☑	☑
Integrated whitelisting/allowlisting	CMTAT Allowlist	☑	☒	☒	☒
External Whitelisting/allowlisting	CMTAT with rule whitelist	☒	☒	☑	☑
RuleEngine / transfer hook	CMTAT with RuleEngine	☒	☒	☑	☑
Upgradibility	CMTAT Upgradeable version	☑	☑	☑	☑
Feepayer/gasless	CMTAT with ERC-2771 module	☑	☒	☒	☑

ForcedBurn/forcedTransfer:

In the standard burn function, it is not possible to burn token from a frozen wallet. CMTAT offers a dedicated function `forcedTransfer` which allows to force a transfer or a burn. If the `forcedTransfer` function is not available, the alternative is to implement only the function `forcedBurn`.

This is what is done for the CMTAT light version which does not include `forcedTransfer`. You can also decide to implement both. In this case, we suggest that only `forcedBurn` can burn tokens and not `forcedTransfer`. With the CMTAT Solidity version, when `forcedTransfer` is available, we do not implement `forcedBurn` to reduce smart contract code size, but this limitation is not necessary present with other blockchains.

Implementation details

Functionalities	CMTAT Solidity	Note
Mint while pause	☑	Dedicated crosschain mint (e.g. `crosschainMint`) cannot be performed while the contract is in the pause state.
Burn while pause	☑	Dedicated crosschain burn (e.g. `crosschainBurn`) cannot be performed while the contract is in the pause state.
Self Burn	☒	Token holder can not burn their own tokens. Only authorised addresses are allowed to burn tokens.
Standard burn on a frozen address	☒	Required to use `forcedTransfer` or `forcedBurn`
Burn tokens with the function `forcedTransfer`	☑	See note above

Self burn

It's deliberate that only the issuer (and not the tokenholder) can burn a token, and that this corresponds to a legal requirement in several countries.

Indeed, once issued, a security can only be cancelled by its issuer, not by its holder. Since the token serves as a vehicle for the security, the same must apply to the token itself. An investor wishing to "get rid of" a token must transfer it to the issuer, who can then cancel it when the law allows.

However, feel free to add it in your CMTAT version if this makes sense for your from a legal or business perspective.

Intellectual property