CMTA Token

> To use the CMTAT, we recommend the latest audited version, from the Releases page.
> Currently, it is the version v2.3.0

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 technical features such as ERC-7802 for cross-chain transfer and upgradeadibility.

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

The CMTAT is an open standard from the Capital Markets and Technology Association (CMTA), which gathers Swiss finance, legal, and technology organizations.
The CMTAT was developed by a working group of CMTA's Technical Committee that includes members from Atpar, Bitcoin Suisse, Blockchain Innovation Group, Hypothekarbank Lenzburg, Lenz & Staehelin, Metaco, Mt Pelerin, SEBA, Swissquote, Sygnum, Taurus and Tezos Foundation.
The design and security of the CMTAT was supported by ABDK, a leading team in smart contract security.

Use case

This reference implementation allows the issuance and management of tokens representing equity securities, and other forms of financial instruments such as debt securities and stablecoin.

CMTAT was initially optimized for the Swiss law framework, but is also suitable for other jurisdictions.

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

Technical

The design and security of the CMTAT was supported by ABDK, a leading team 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.

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

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:

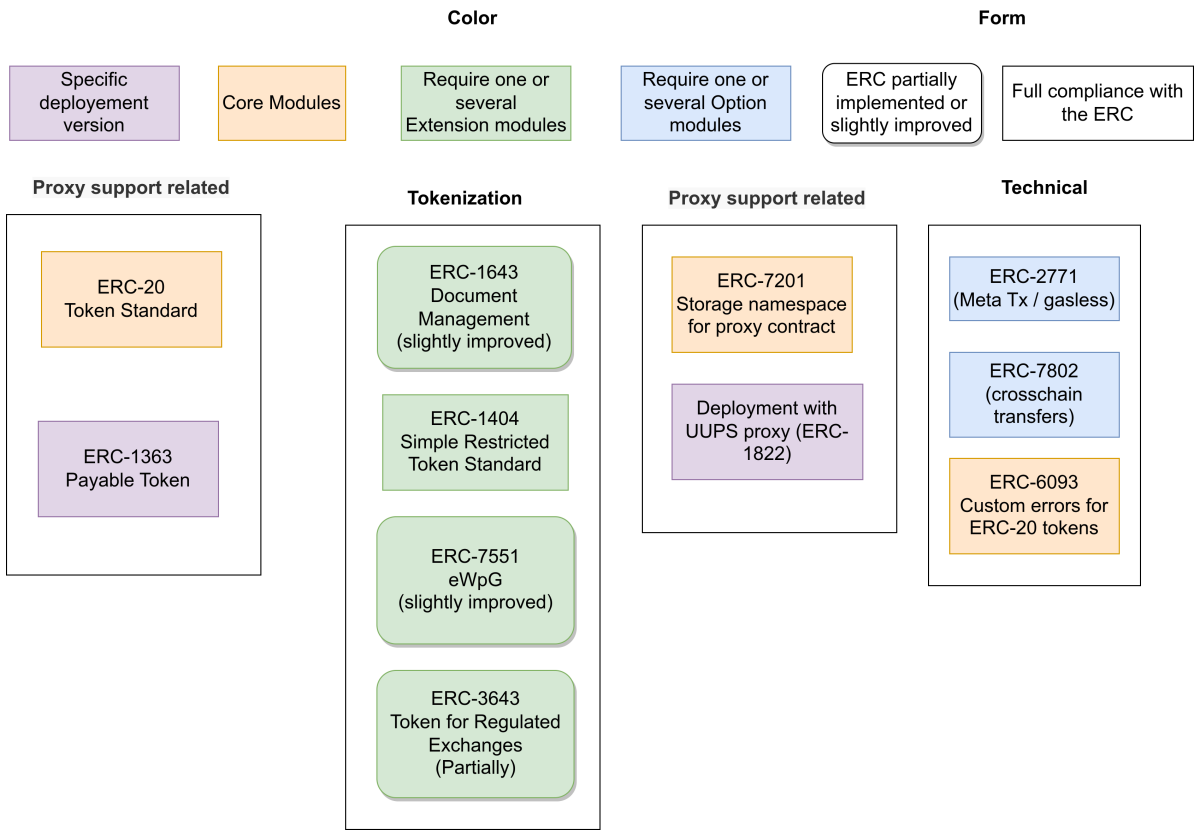
- Transfer restriction through allowlisting/whitelisting (can be also done with a `RuleEngine`)
- Put Debt information and Credit Events on-chain
- Cross-chain functionalities with ERC-7802
- "Gasless" (MetaTx) transactions with ERC-2771

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 1.0	CMTAT 2.30	CMTAT 3.0.0					
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)	☒	☑	☑	☑	☑

	Associated contracts/modules	ERC status	CMTAT 1.0	CMTAT 2.30	CMTAT 3.0.0					
ERC-3643 (Without on-chain identity)	Core + ERC20EnforcementModule (extensions)	Standard Track (final)	☒	☒	☑	☒	☑	☑	☑	☑
ERC-7551 (Slightly improved)	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										
ERC-2771 (Meta Tx / gasless)	MetaTxModule (options)	Standard Track (final)	☑	☑	☑	☒	☑	☑	☑	☒
ERC-6093 (Custom errors for ERC-20 tokens)	-	Standard Track (final)	☒	☒	☑	☑	☑	☑	☑	☑
ERC-7802 (cross-chain token/transfers)	ERC20CrossChainModule (options)	Draft	☒	☒	☑	☒	☑	☑	☒	☒

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 several same functions. The interface is available in IERC3643Partial.sol

To represent the level of similarity between ERC-3643 interface and CMTAT functionnalities, we have created three levels of conformity.

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)

// IERC3643Base
// BaseModule
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 batchFreezePartialTokens(address[] calldata _userAddresses, uint256[]
calldata _amounts) external;
function batchUnfreezePartialTokens(address[] calldata _userAddresses,
uint256[] calldata _amounts) 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 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 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 term of access control.

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

Pause

Module: PauseModule

ERC-3643	**CMTAT 3.0.0**	**Deployment version**
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 3.0**	**All deployment version**

ERC-3643	CMTAT 3.0	All deployment version
`setName(string calldata _name) external;`	`setName(string calldata name_)`	☑
`setSymbol(string calldata _symbol) external`	`setSymbol(string calldata symbol_)`	☑

Supply Management (burn/mint)

ERC-3643	CMTAT 3.0 Modules	CMTAT 3.0 Functions	Deployment version
`batchMint(address[] calldata _toList, uint256[] calldata _amounts) external;`	ERC20MintModule	`mint(address account, uint256 value)`	All
`batchMint(address[] calldata _toList, uint256[] calldata _amounts) external;`	ERC20MintModule	`batchMint(address[] calldata accounts,uint256[] calldata values)`	All
`function 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 3.0	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

ERC-3643	CMTAT 3.0	Deployment version
`batchForcedTransfer(address[] calldata _fromList, address[] calldata _toList, uint256[] calldata _amounts) external`	Not implemented	-

ValidationModuleC

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

ERC-3643	CMTAT 3.0	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
>
> Status: draft

This section presents a correspondence table between ERC-7551 and their equivalent functions inside CMTAT.

The ERC-7551 is currently a draft ERC proposed by the Federal Association of Crypto Registrars from Germany to tokenize assets in compliance with eWPG.

The interface is supposed to work on top of additional standards that cover the actual storage of ownership of shares of a security in the form of a token (e.g. ERC-20 or ERC-1155).

CMTAT modification

Since it is not yet an official standard, we decided to use the same name and signature as ERC-3643. Typically, we define a function `burn` instead of `destroyTokens`.

The implemented interface is available in IERC7551

N°	Functionalities	ERC-7551 Functions	CMTAT 3.0.0	Implementations details	Modules
1	Freeze and unfreeze a specific amount of tokens	freezeTokens unfreezeTokens	☑	Implement ERC-3643 function `freezePartialTokens` and `unfreezePartialTokens` (with and without a `data` parameter) + ERC-3643 function `setAddressFrozen` (with and without a `data` parameter)	EnforcementModule ERC20EnforcementModule
2	Pausing transfers The operator can pause and unpause transfers	pauseTransfers	☑	Implement ERC-3643 functions `pause/unpause` + `deactivateContract`	PauseModule

N°	Functionalities	ERC-7551 Functions	CMTAT 3.0.0	Implementations details	Modules
3	Link to off-chain document Add the hash of a document	setPaperContractHash	Equivalent functionality	The hash can be put in the field `Terms` Terms is represented as a Document (name, uri, hash, last on-chain modification date) based on ERC-1643	ExtraInformationModule
4	Metadata JSON file	setMetaDataJSON	☑	Define function `setMetaData`	ExtraInformationModule
5	Forced transfersTransfer `amount` tokens to `to` without requiring the consent of `fro`m	forceTransferFrom	☑	Implement ERC-3643 function `forcedTransfer` (with and without a `data` parameter)	ERC20EnforcementModule
6	Token supply managementreduce the balance of `tokenHolder` by `amount` without increasing the amount of tokens of any other holder	destroyTokens	☑	Implement ERC-3643 function `burn` / `batchBurn` (with and without a `data` parameter)	BurnModule
7	Token supply managementincrease the balance of `to` by `amount` without decreasing the amount of tokens from any other holder.	issue	☑	Implement ERC-3643 function `mint` / `batchMint` (with and without a `data` parameter)	MintModule
8	Transfer compliance Check if a transfer is valid	`canTransfer()` and a `canTransferFrom()`	☑	Implement ERC-3643 function `canTransfer` as well as a specific function `canTransferFrom`	ValidationModuleCore

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


// IERC7551Base
// ExtraInformationModule
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 extensible 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 here: Cross-Chain bridge support

Deployment version: since it is an extension 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: module and engines

The main schema describing the architecture can be found here: architecture.pdf

Base contract

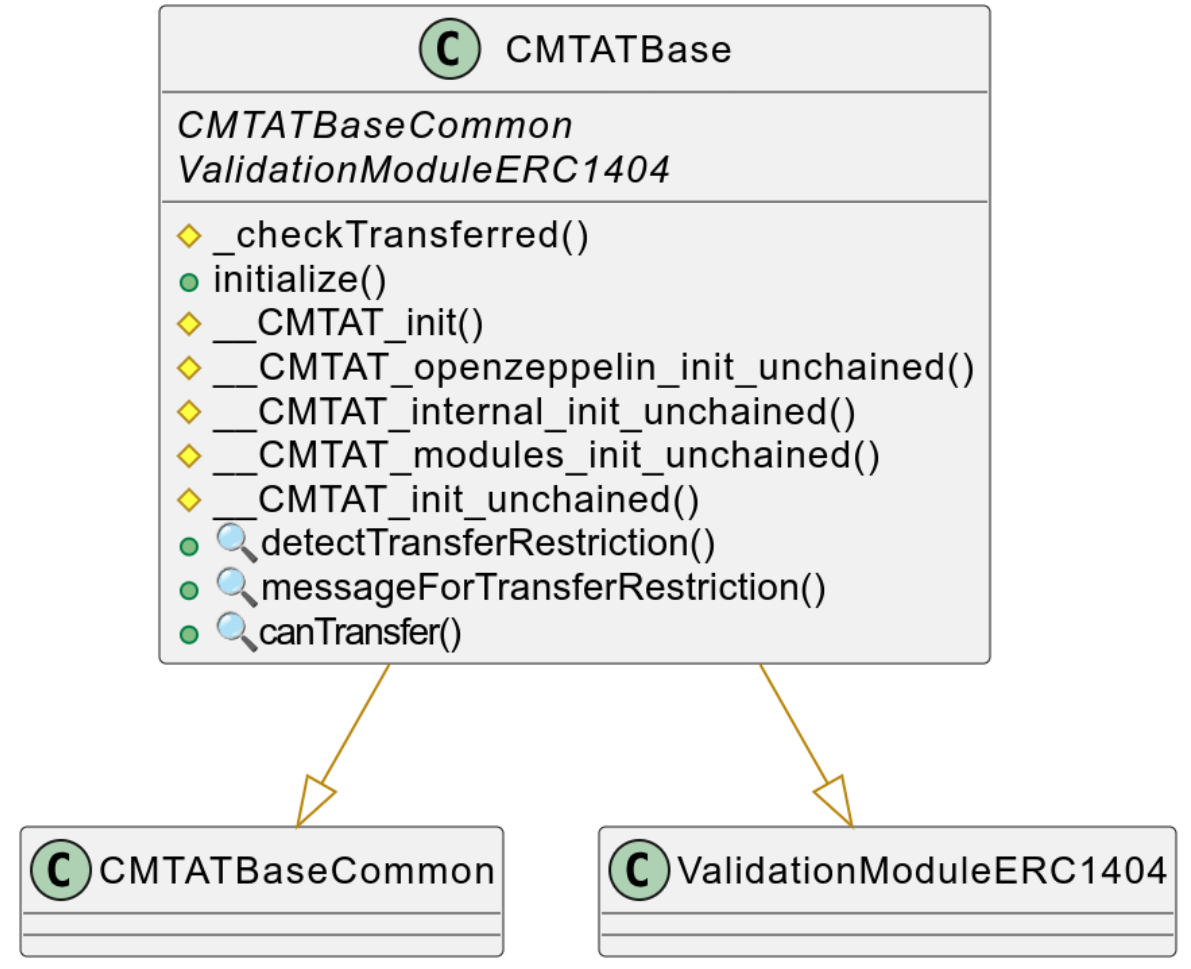
The base contracts are abstract contracts, so not directly deployable, which inherits from several different modules.

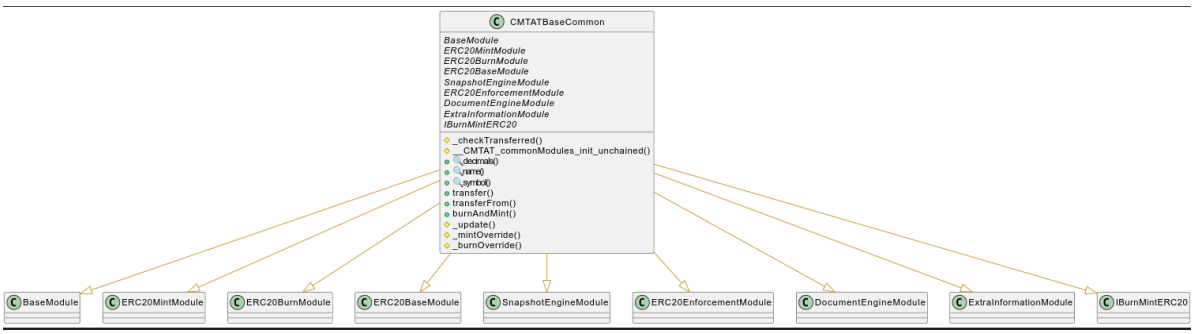
Base contracts are used by the different deployable contracts (CMTATStandalone, CMTATUpgradeable,...) to inherits from the different modules

Name	Description	Associated contracts deployments
CMTATBase	Inherit from all core and extensions modules	CMTAT Standalone / Upgradeable CMTAT Upgradeable UUPS
CMTATBaseCore	Inherit from from all core modules	CMTAT Light (Upgradeadble & Standalone
CMTATBaseGeneric	Inherits from non-ERC20 related modules	- (Only mock available)

Name	Description	Associated contracts deployments
CMTATBaseERC1363	Inherit from CMTATBase, but also ERC-1363 OpenZeppelin contract and MetaTxModule (ERC-2771)	CMTAT ERC1363 (Upgradeable & Standalone)
CMTATBaseOption	Inherit from CMTATBase, but also from several other option modules	CMTAT Standalone / Upgradeable
CMTATBaseAllowlist	Inherit from CMTATBaseCommon, but also from ValidationModuleAllowlist	CMTAT Allowlist (upgradeable & Standalone)

CMTATBase





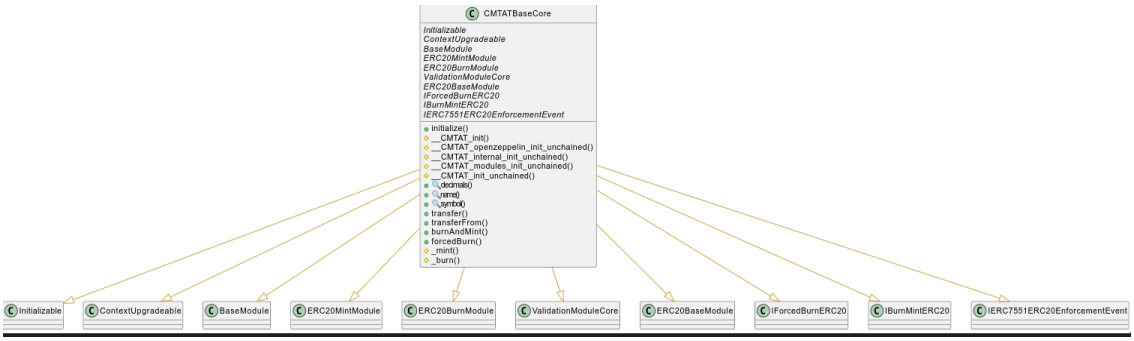
CMTAT Base adds several functions:

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

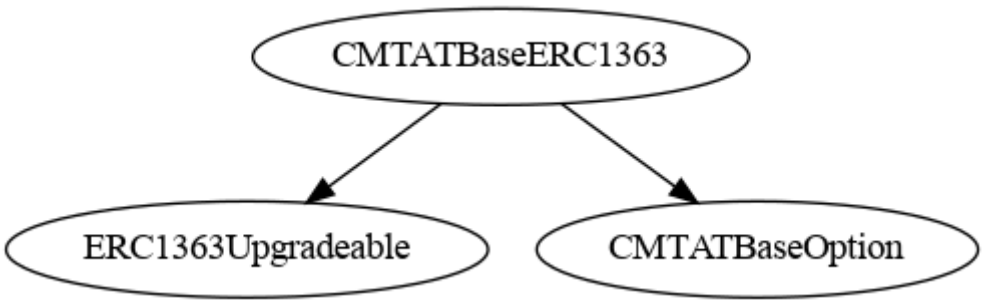
CMTAT Base Core

CMTAT Base Core adds several functions:

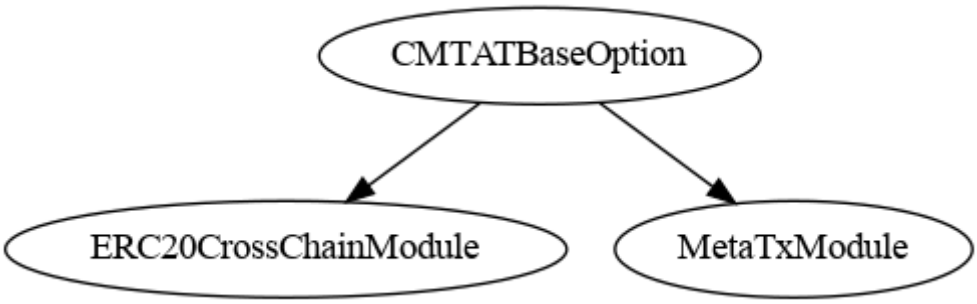
- `burnAndMint` to burn and mint atomically in the same function.
- `forcedBurn` to allow the admin to burn tokens from a frozen address (defined in EnforcementModule)
    - This function is not required in CMTATBase because the function `forcedTransfer` (ERC20EnforcementModule) can be used instead.



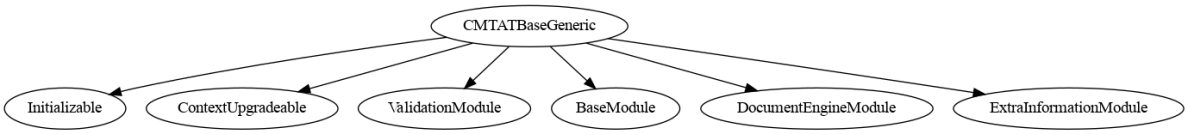
CMTAT ERC1363 Base



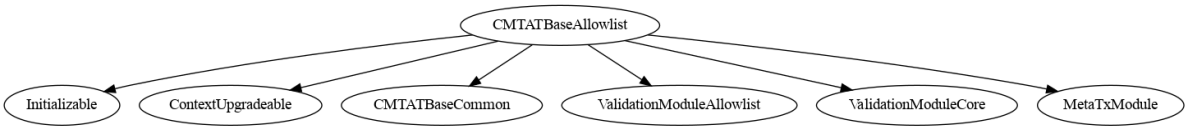
CMTAT Base Option



CMTAT Base Generic



CMTAT Base Allowlist



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 require to perform a security audit 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
    - **Extension** (Wrapper sub-category): not required to be CMTA compliant, "bonus features" (snapshotModule, debtModule)
    - **Options**: also bonus feature to meet specific use case through specific deployment version.

List

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

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

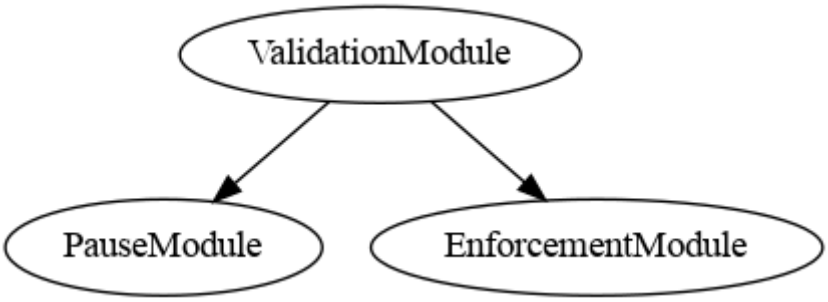
- "fn" means function
- Changes made in a release are considered maintained in the following release unless explicitly stated otherwise

Controllers

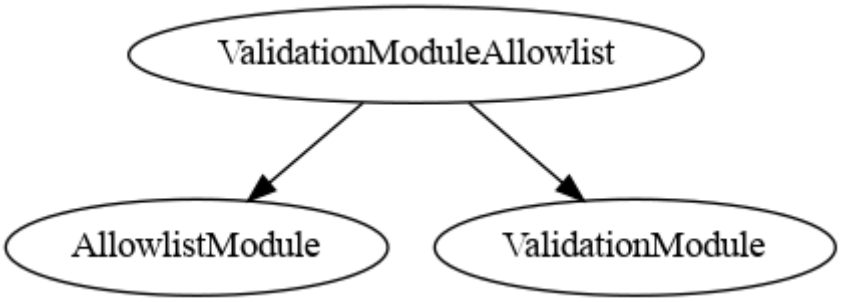
Modules	Type	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT 3.0.0		
Deployment version						Standalone, Upgradeable, UUPS, Debt	CMTAT Allowlist	CMTAT Light
ValidationModule	Controllers	Check transfer validity by calling the Pause and Enforcement modules	ValidationModule.sol	☑	☑	☑	☑	☑
ValidationModuleAllowlist	Controllers	Check transfer validity by calling Allowlist module	ValidationModuleAllowlist.sol	☒	☒	☒ Only CMTAT Allowlist)	☑	☒
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	☑	☑	☑	☑	☑
ValidationModuleRuleEngine	Extensions	Set and call the ruleEngine to check transfer.	ValidationModuleRuleEngine.sol	☑	☑	☑	☒	☒
ValidationModuleERC1404	Extensions	Implements ERC-1404	ValidationModuleERC1404.sol	☑	☑	☑	☒	☒

Controllers

- ValidationModule

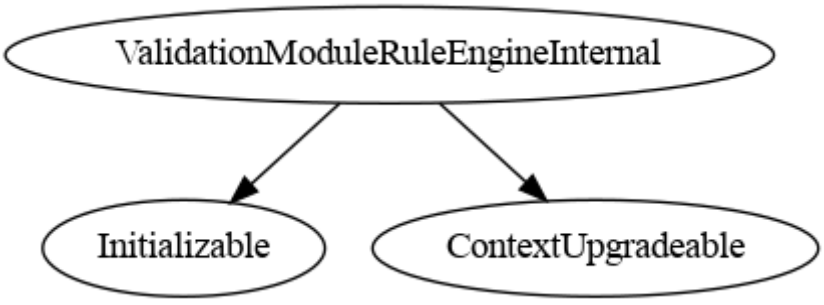


- ValidationModuleAllowlist



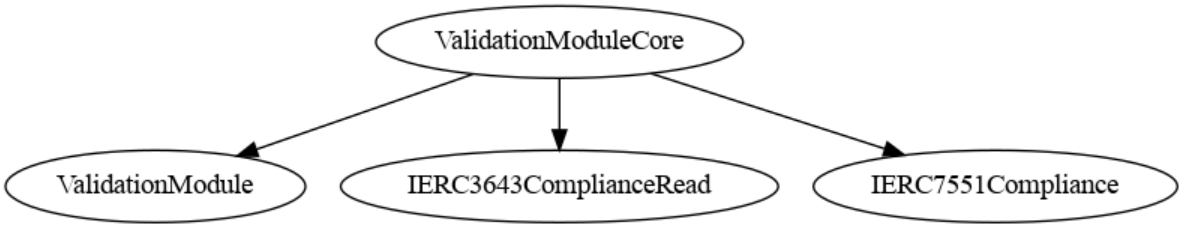
Internal

- ValidationModuleRuleEngineInternal



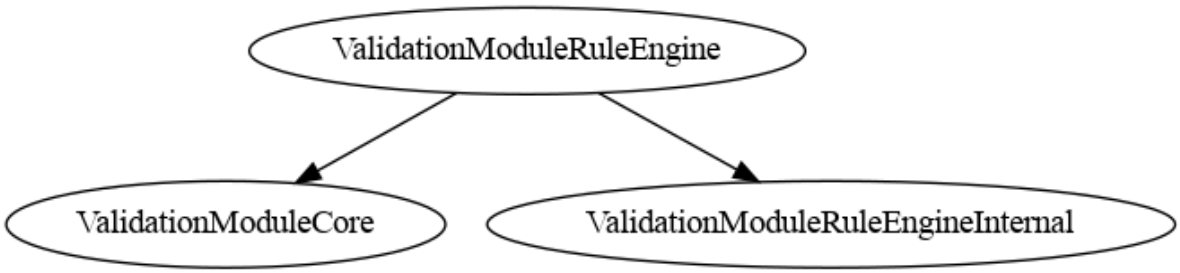
Core

- ValidationModuleCore

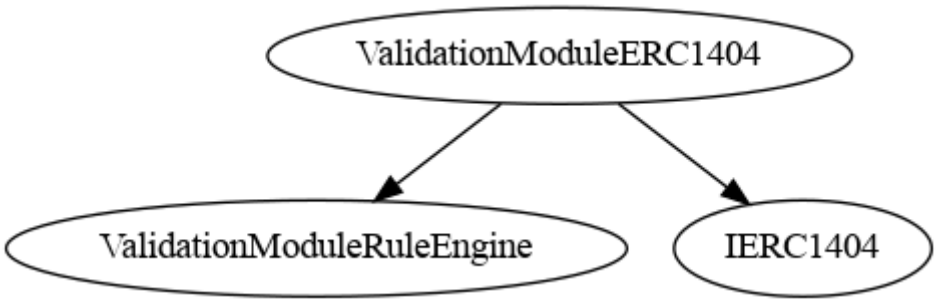


Extensions

- 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
BaseModule	Contract version	BaseModule.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`
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`
Enforcement	Freeze/unfreeze address	EnforcementModule.sol	☑	☑	☑

Modules	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT 3.0.0
ERC20Base	decimals, set name & symbo	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 2.30	CMTAT 3.0.0		
Deployment version					Standalone & Upgradeable	Allowlist	Debt
ERC20CrossChain	Cross-chain functions (ERC-7802)	ERC20CrossChainModule.sol	☒	☒	☑	☒	☒
DebtModule	Set Debt Info	DebtModule.sol	☒	☑	☒	☒	☑ (Don't include CreditEvents managed by DebtEngineModule)

Modules	Description	File	CMTAT 1.0	CMTAT 2.30	CMTAT 3.0.0		
DebtEngineModule	Add a DebtEngine module (requires to set CreditEvents)	DebtEngineModule.sol	☒	☒	☒	☒	☑
MetaTx	ERC-2771 support	MetaTxModule.sol	☑	☑ (forwarder immutable)	☑	☒	☒
Allowlist	Add integrated allowlist support	Allowlist.sol	☒	☒	☒	☑	☒

Security

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

Access Control (RBAC)

CMTAT uses a RBAC access control by using the contract `AccessControl` from OpenZeppelin.

Each modules define the roles useful to restricts its functions.

By default, the `admin` has all the roles and this behavior is defined in the `AuthorizationModule` by overriding the function `hasRole`.

See also docs.openzeppelin.com - AccessControl

Role list

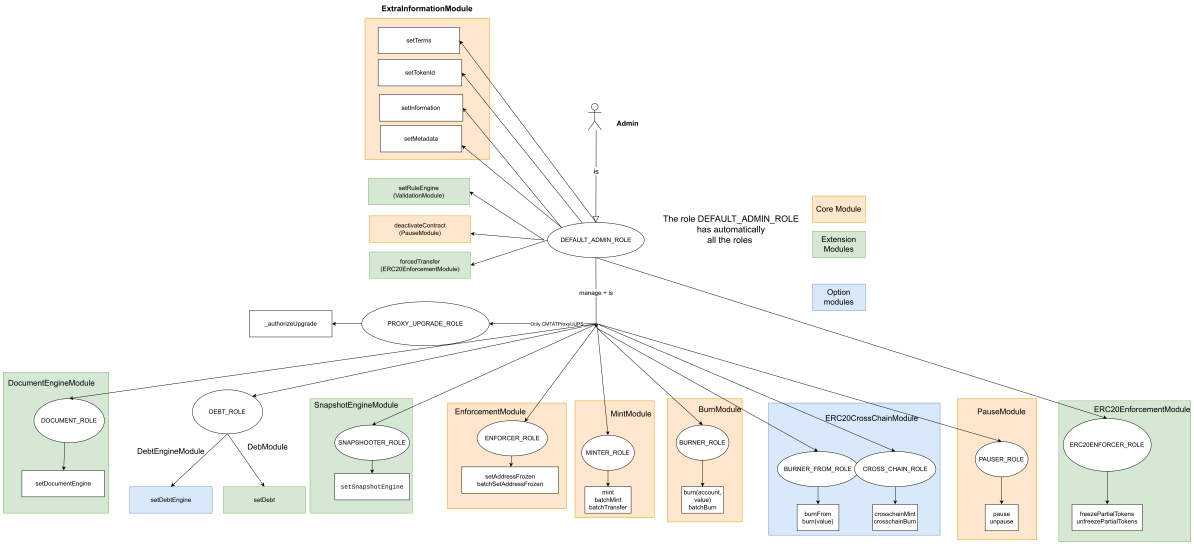
Here the list of roles and their 32 bytes identifier.

	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
ERC20ENFORCER_ROLE	ERC20EnforcementModule	0xd62f75bf68b069bc8e2abd495a949fafec67a4e5a5b7cb36aedf0dd51eec7e72

	Defined in	32 bytes identifier
Option Modules		
CROSS_CHAIN_ROLE	ERC20CrossChainModule	0x620d362b92b6ef580d4e86c5675d679fe08d31dff47b72f281959a4eecdd036a
BURNER_FROM_ROLE	ERC20CrossChainModule	0x5bfe08abba057c54e6a28bce27ce8c53eb21d7a94376a70d475b5dee60b6c4e2
ALLOWLIST_ROLE	AllowlistModule	0x26a560d834a19637eccba4611bbc09fb32970bb627da0a70f14f83fdc9822cbc
DEBT_ROLE	DebtModule	0xc6f3350ab30f55ce45863160fc345c1663d4633fe7cacfd3b9bbb6420a9147f8

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.

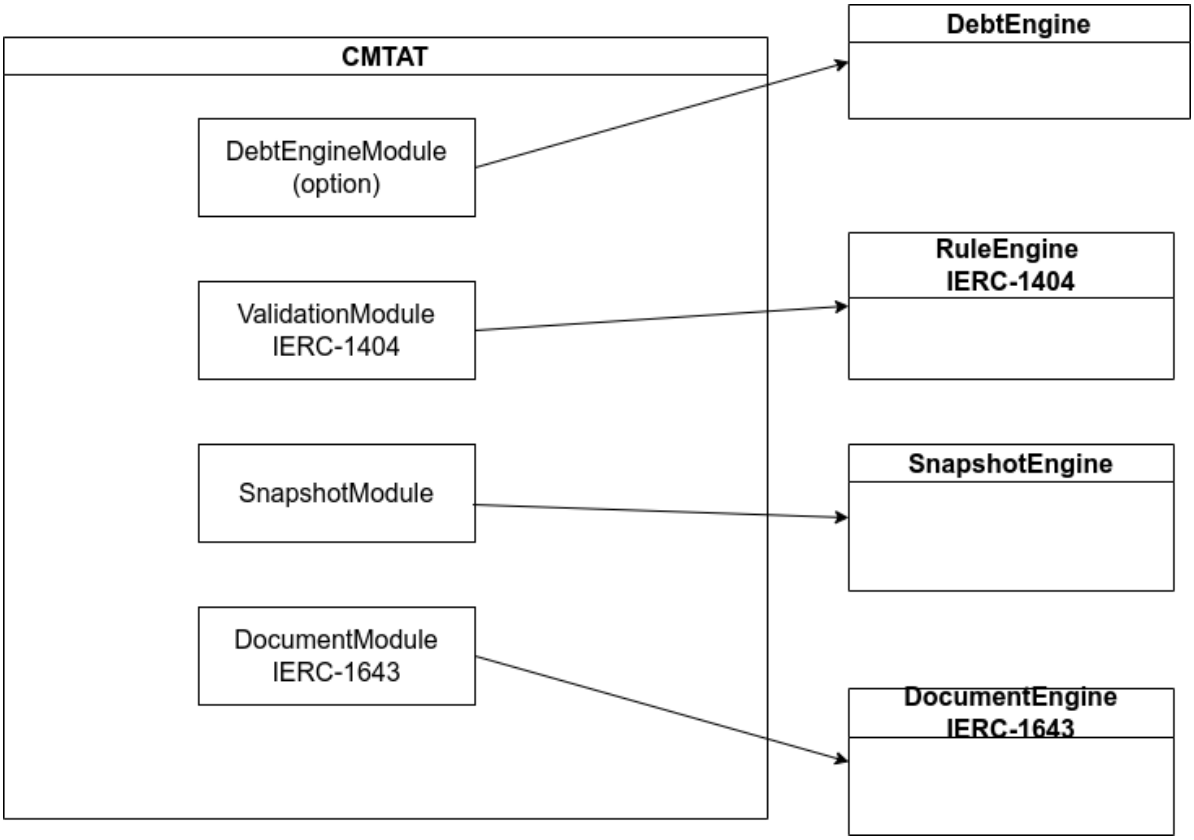
The new admin can also revoke role from the current/old admin by calling `revokeRole`.

It is also possible to have several different admin.

Engines

Engines are external smart contracts called by CMTAT modules.

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



RuleEngine (IERC-1404)

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

This contract is defined in the `ValidationModule`.

An example of RuleEngine is also available on GitHub.

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

CMTAT version	RuleEngine
CMTAT v3.0.0	
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.

A possible rule is a whitelist rule where only the address inside the whitelist can perform a transfer

Requirement

Since the version 2.4.0, the requirement to use a RuleEngine are the following:

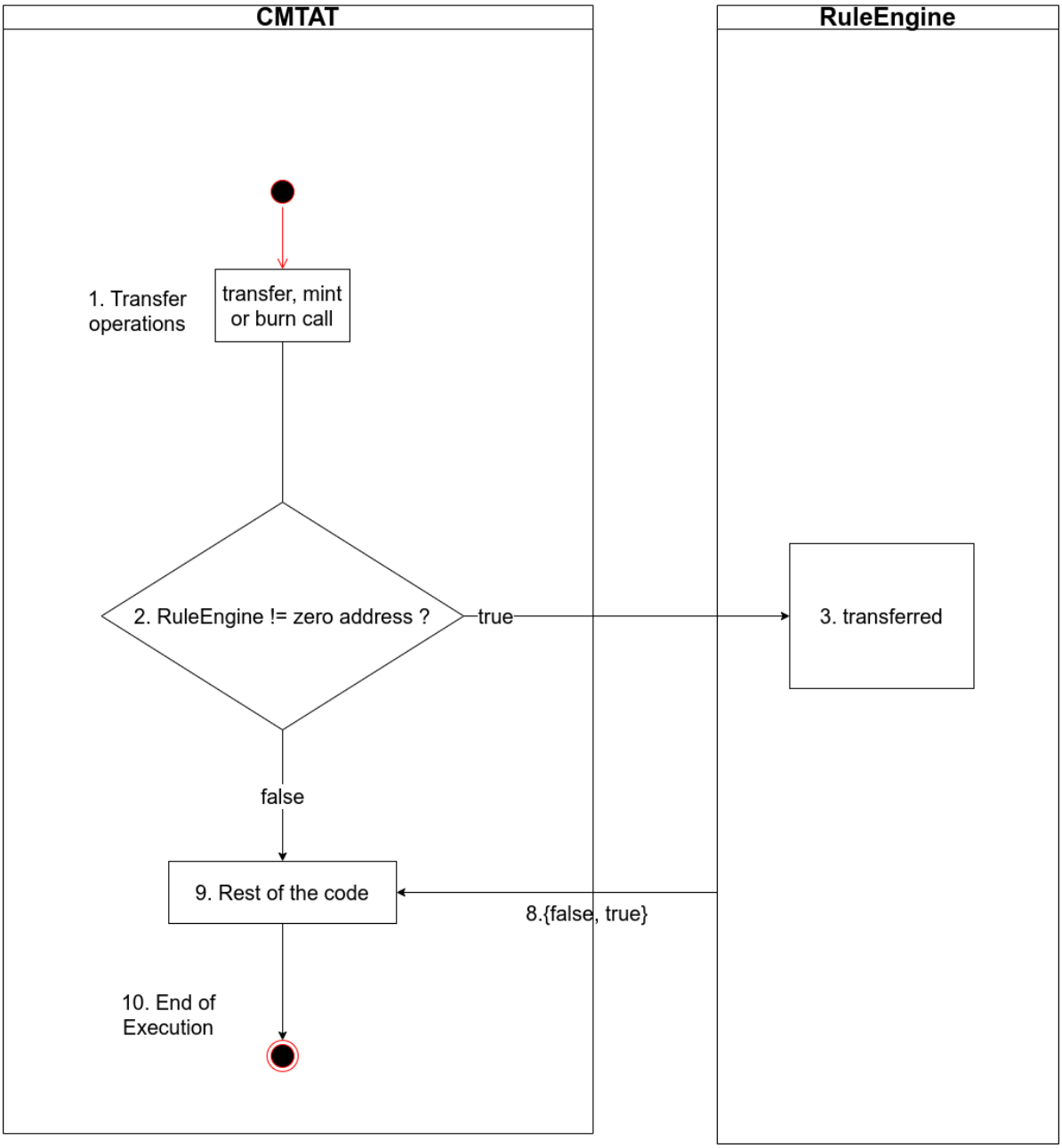
> The `RuleEngine` must import and implement the interface `IRuleEngine` which declares the function `transferred` and `canApprove` with several other functions related to IERC1404.

This interface can be found in interfaces/engine/IRuleEngine.sol.

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

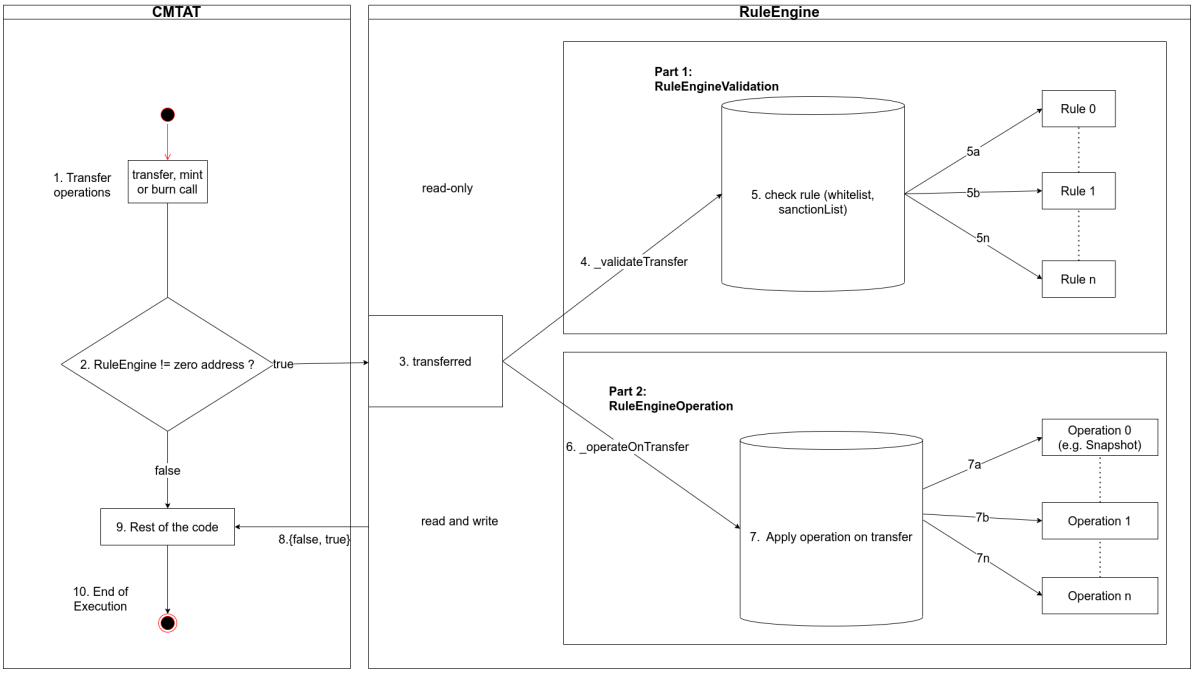
Before each transfer, the CMTAT calls the function `transferred` which is the entrypoint for the RuleEngine.

Further reading: [Taurus - Token Transfer Management: How to Apply Restrictions with CMTAT and ERC-1404](#) (version used CMTAT v2.4.0)



Example of a CMTAT using the CMTA ruleEngine:

In this example, the token holder calls the function `transfer` which triggers a call to the `RuleEngine` and the different rules associated.



SnapshotEngine

Engine 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 {
    /**
     * @dev Returns true if the operation is a success, and false otherwise.
     */
    function operateOnTransfer(address from, address to, uint256 balanceFrom,
uint256 balanceTo, uint256 totalSupply) external;
}
```

CMTAT	SnapshotEngine
CMTAT v3.0.0	
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	SnapshotEngine
CMTAT v1.0.0	Include inside SnapshotModule, but not gas efficient (audited)

DebtEngine

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

- It defined in the `DebtEngineModule` (option module)
- It extends the `DebtModule` (option module) by allowing to set Credit Events while the DebtModule only allows to set debt info.
- If a `DebtEngine` is configured, the function `debt`() will return the debt configured by the RuleEngine instead of the `DebtModule`.

CMTAT only implements two functions, available in the interface IDebtEngine to get information from the debtEngine.

```
interface IDebtEngine is IDebtEngine {
    function debt() external view returns(IDebtGlobal.DebtBase memory);
    function creditEvents() external view returns(IDebtGlobal.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	
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 document 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`)
- 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.

Use an external contract provides two advantages:

- Reduce code size of CMTAT, which is near of the maximal size limit
- Allow to manage documents 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	
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 check 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 function which allows 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;
}
```

Gasless support (ERC-2771 / MetaTx module)

The CMTAT supports client-side gasless transactions using the 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` while allowing upgrades on the main contract (see *Deployment via a proxy* above).

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 storage of the proxy.

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 transfer as well as burn/mint operation

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

- 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;
}
```

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 upgrate (rolled out in Q1 of 2024) has removed support for SELFDESTRUCT (see EIP-6780).

The `kill()` function will therefore not behave as it was used, and we have replaced it by the function `deactivateContract` .

How it works

This function sets a boolean state variable `isDeactivated` to true and puts the contract in the pause state.
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)

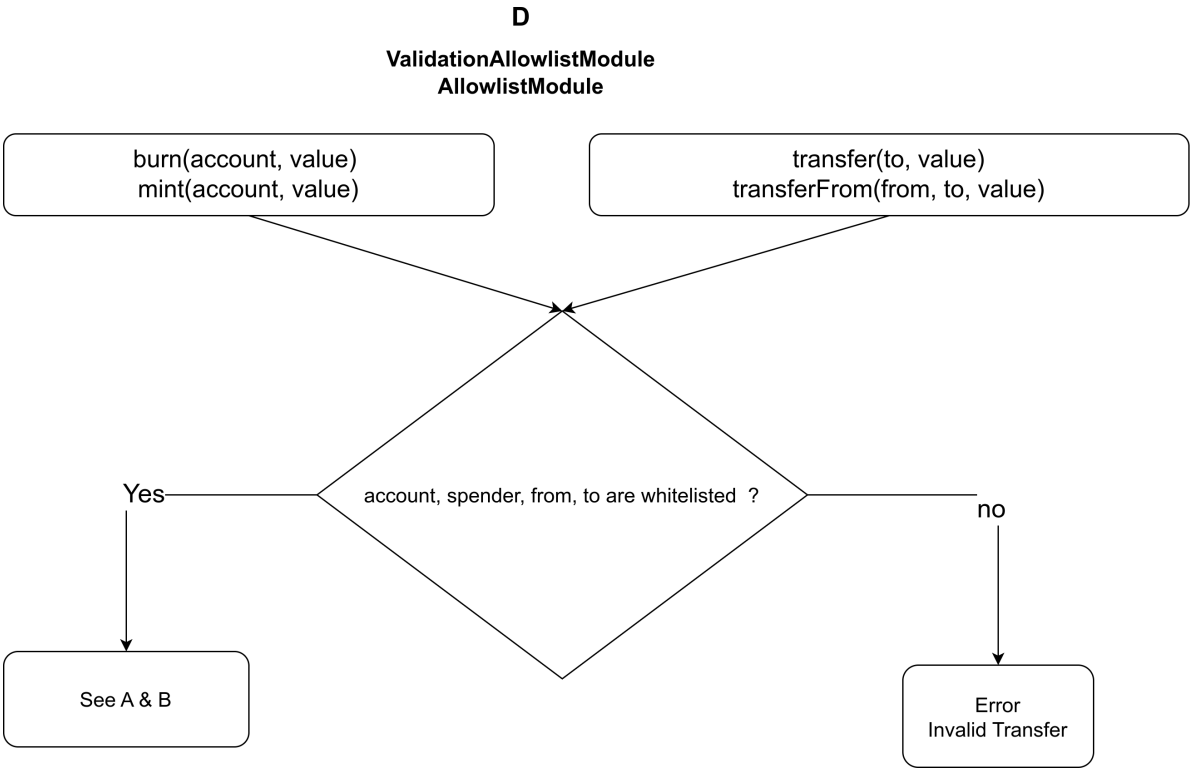
	burn	**batchBurn**	**burnFrom**	**mint**	**batchMint**	**crosschain burn**	**Crosschain mint**	**forcedTransfer**
Module	ERC20Burn	ERC20Burn	ERC20CrossChain	ERC20Mint	ERC20Mint	ERC20CrossChain	ERC20CrossChain	ERC20Enforcement
Module type	Core	Core	Options	Core	Core	Options	Options	Extensions
Allow operation on a frozen address	☒	☒	☒	☒	☒	☒	☒	☑
Unfreeze missing funds if active balance is not enough (ERC20EnforcementModule)	☒	☒	☒	-	-	☒	-	☑
Call the RuleEngine	☑	☑	☑	☑	☑	☑	☑	☒

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 IAllowlist {
    event AddressAddedToAllowlist(address indexed account, bool indexed status,
address indexed enforcer, bytes data);
    event AllowlistEnableStatus(address indexed operator, bool status);
    /**
    * @notice return true if `account`is in the allowlist, false otherwise
    */
    function isAllowlisted(address account) external view virtual returns
(bool);
    /**
     * @notice add/remove an address to/from the allowlist
     */
    function setAddressAllowlist(address account, bool status) external
virtual;

    /**
```

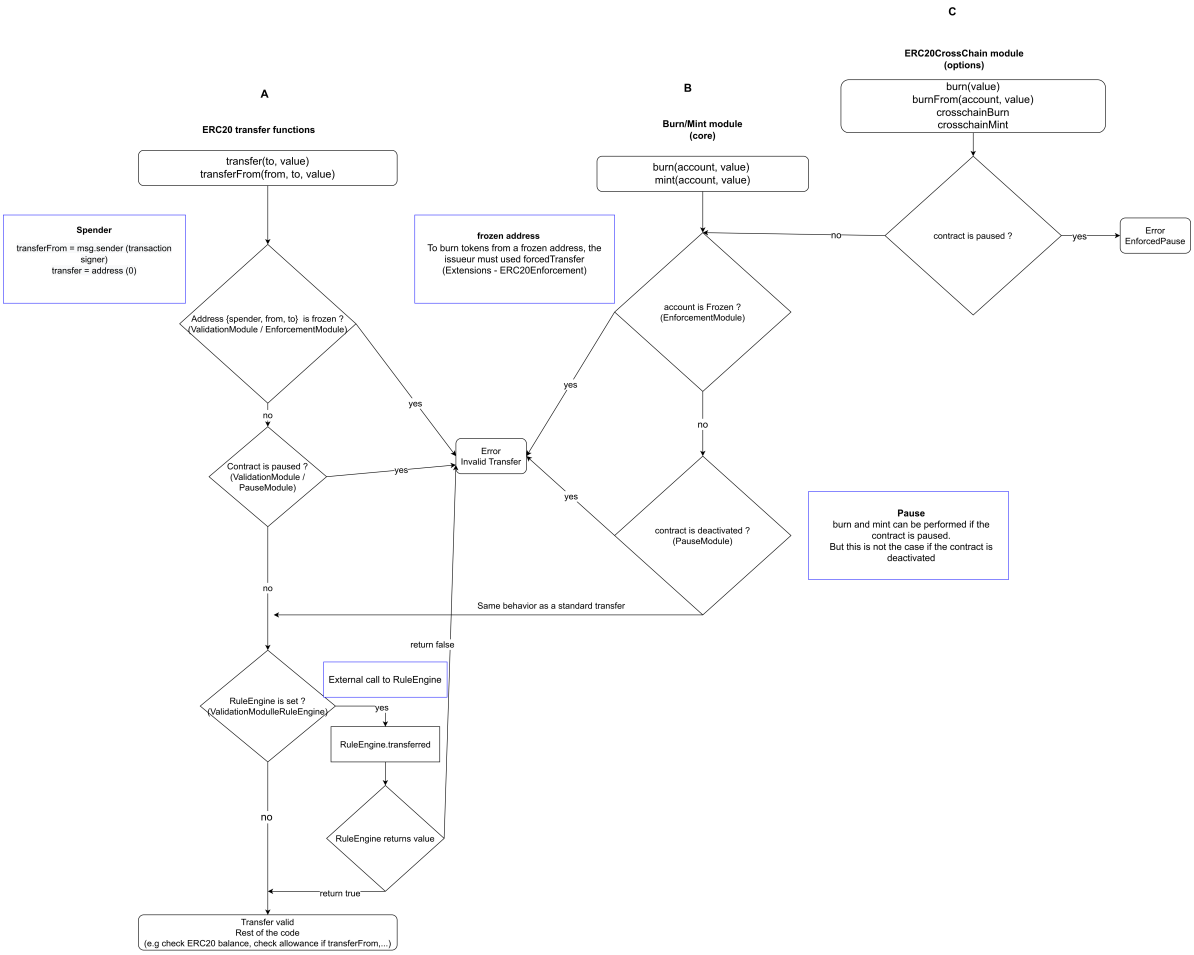
```
     * @notice add/remove an address to/from the allowlist
    */
    function setAddressAllowlist(address account, bool status, bytes calldata
data) external virtual
    /**
    * @notice Batch version of {setAddressAllowlist}
    */
    function batchSetAddressAllowlist(address[] calldata accounts, bool[]
calldata status) external virtual
    /**
    * @notice enable/disable allowlist
    */
    function enableAllowlist(bool status) external virtual

    /**
    * @notice Returns true if the list is enabled, false otherwise
    */
    function isAllowlistEnabled() external view virtual returns (bool)
}
```



Schema

Here a schema describing the different check performed during:

- transfer & transferFrom
- burn / mint (supply management)
- burn / mint for crosschain transfers



Supply management

Event

Name	Defined	Stdanard	Concerned functions
Transfer(address indexed from, address indexed to, uint256 value);	IERC20 (OpenZeppelin)	ERC-20	All functions which impacts the supply because a burn/mint is a transfer
Mint(address indexed account, uint256 value, bytes data);	IERC7551Mint	ERC-7551 (draft standard)	mint (ERC20MintModule)
BatchMint(address indexed minter, address[] accounts, uint256[] values		-	BatchMint (ERC20MintModule)
Burn(address indexed account, uint256 value, bytes data);	IERC7551Burn	ERC-7551 (draft standard)	burn (ERC20BurnModule)

Name	Defined	Stdanard	Concerned functions
BatchBurn(address indexed burner, address[] accounts, uint256[] values)		-	BatchMint (ERC20BurnModule)
BurnFrom(address indexed burner, address indexed account, address indexed spender, uint256 value);	IBurnERC20	-	brunFrom (ERC20CrossChain)
CrosschainMint(address indexed to, uint256 value, address indexed sender)	IERC7551	ERC-7551	crosschainMint (ERC20CrossChain)
CrosschainBurn(address indexed from, uint256 value, address indexed sender)	IERC7551	ERC-7551	crosschainBint (ERC20CrossChain)

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)

Option module

BurnFrom

```
interface IBurnFromERC20 {
  event BurnFrom(address indexed account, address indexed spender, uint256
value);
  function burnFrom(address indexed burner, address indexed account, uint256
value) external;
}
```

ERC-7802

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

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`

Deployment model

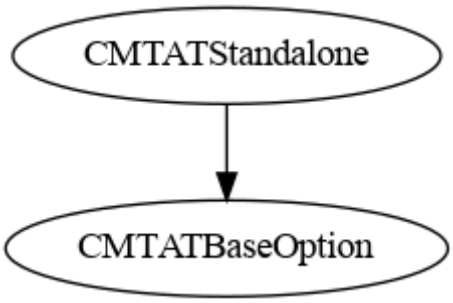
Contracts for deployment are available in the directory contracts/deployment

CMTAT Model	Description	Standalone/Proxy	Contract	Remark
CMTAT Standard	Deployment without proxy (immutable)	Standalone	CMTATStandalone	Core & extension module without Debt, Allowlist, ERC-3643 and UUPS Include also two option modules: ERC20Crosschain & MetaTx
	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 ERC-3643 support
	-	Upgradeable	CMTATUpgradeableERC1363	
Light	Only core modules	Standalone	CMTATStandaloneLight	-
		Upgradeable	CMTATUpgradeableLight	
Debt	Set Det information and CreditEvents (through DebtEngine)	Standalone	CMTATStandaloneDebt	Add the debt support. Contrary to the standard version, it does not include the modules MetaTx and ERC20Crosschain
		Upgradeable	CMTATUpgradeableDebt	-
Allowlist	Restrict transfer to an allowlist (whitelist)	Standalone	CMTATStandaloneAllowlist	Contrary to the standard version, it does not include ERC-1404 support (ValidationModuleERC1404) & ERC20Crosschain
		Upgradeable	CMTATUpgradeableAllowlist	

Standard Standalone

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

Here 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.

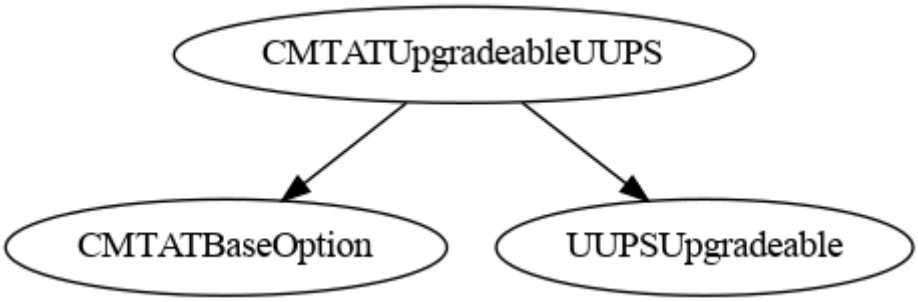
- The implementation contract to use with a TransparentProxy is the `CMTATUpgradeable`.
- The implementation contract to use with a UUPSProxy is the `CMTATUpgradeableUUPS`.

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

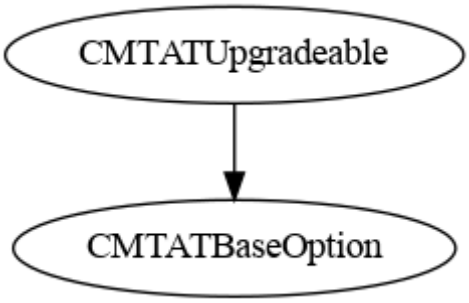
See the OpenZeppelin Upgrades plugins for more information about plugin upgrades in general.

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

Initialize functions

For wrapper modules, we have removed the public function `{ContractName}_init` to reduce the size of the contracts since inside the public initializer function to initialize your proxy, you have to call the difference 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.

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

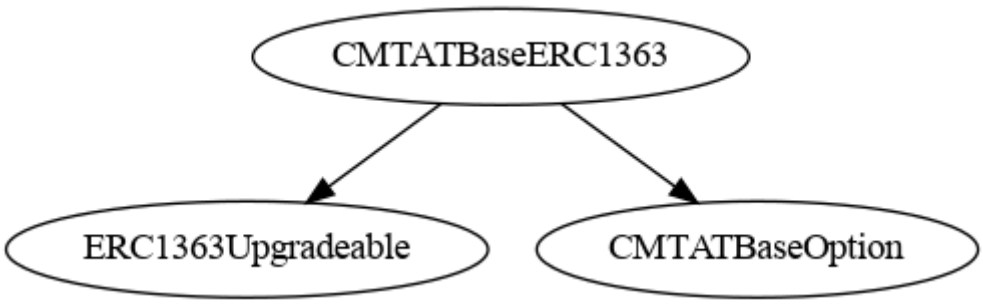
ERC-1363 is an extension interface for ERC-20 tokens that supports executing code on a recipient contract after transfers, or code on a spender contract after approvals, in a single transaction.

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

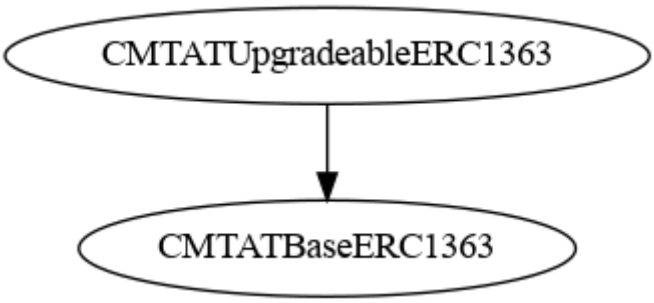
More information on this standard here: erc1363.org, RareSkills - ERC-1363

Inheritance

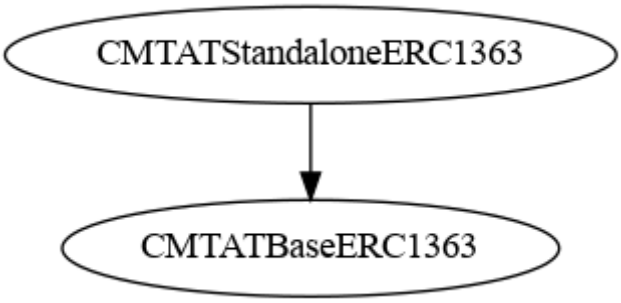
- CMTAT ERC-1363 Base



- CMTAT Upgradeable ERC-1363



- CMTAT Standalone ERC-1363

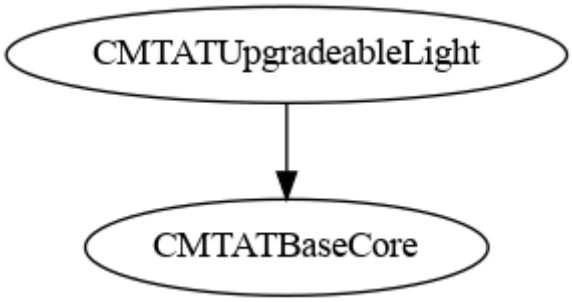


Light version

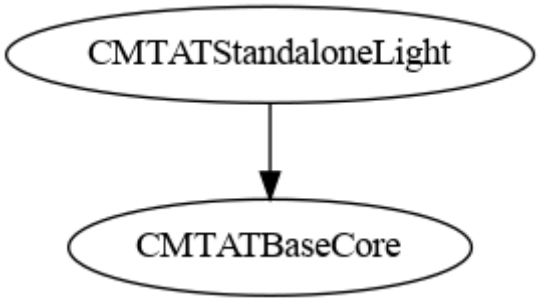
The light version only includes core modules.

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 version which includes the extension module `ERC20EnforcementModule` because this modules contains a function `forcedTransfer` which can be used instead.

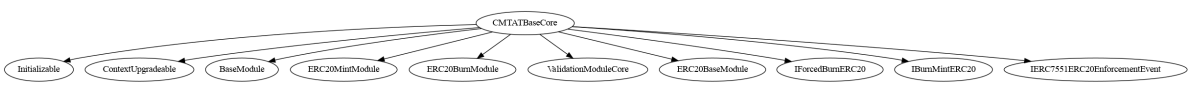
- CMTAT Upgradeable Light



- CMTAT Standalone Light



- CMTATBaseCore



Debt version

This deployment version includes the optional module `DebtModule` and `DebtEngineModule` which allows to store information related to the debt instrument inside the smart contract, as well as related `Credit Events` through an external engine called `DebtEngine`.

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

Struct

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

Field name	Type	Description

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 interfaces/tokenization/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
dayCountConvention	string	-
businessDayConvention	string	-
currency	string	-

Field name	Type	Description
currencyContract	address	-

Credits events

Defined by the struct `CreditEvents` in interfaces/tokenization/ICMTAT.sol.

Contrary to the debt information, it requires the external contract `DebtEngine` to set the information

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

	Type
flagDefault	bool
flagRedeemed	bool
rating	string

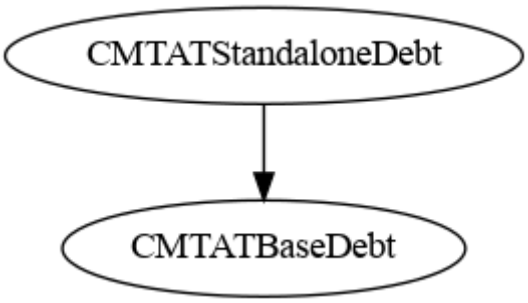
Specification

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

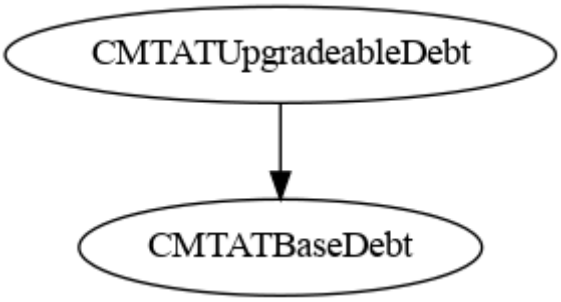
	Module	Read/get function	Write/set functions	Require DebtEngine	Internal field
Debt Identifier	DebtModule/ DebtEngineModule	debt()	setDebt(...)	☒ (but can be used)	`_debt`
Debt Instrument	DebtModule DebtEngineModule	debt()	setDebt(...) setDebtInstrument(...)	☒ (but can be used)	`_debt`
Credit Events	DebtEngineModule	creditEvents()	- (require `DebtEngine`)	☑	- (stores by the `DebtEngine`)

Schema

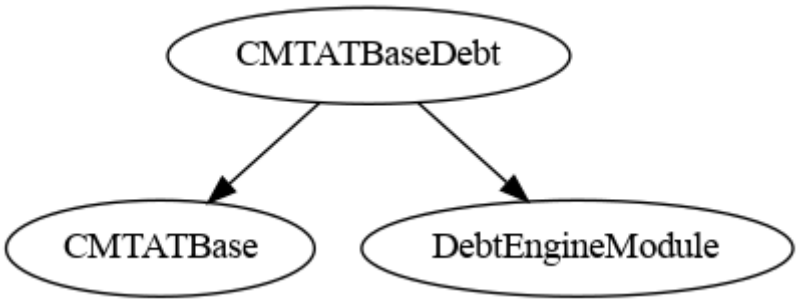
- CMTAT Standalone Debt



- CMTAT Upgradeable Debt

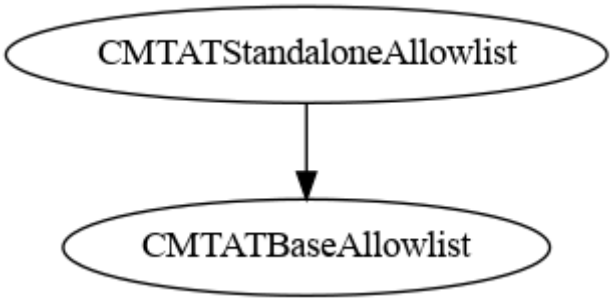


- CMTAT Base Debt

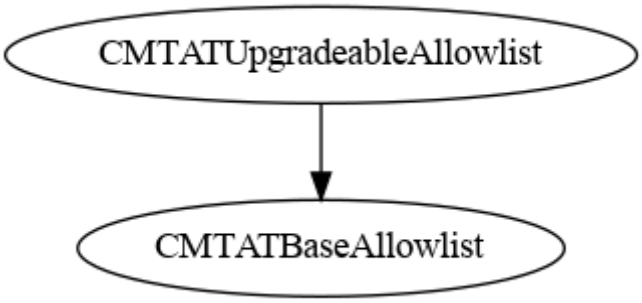


Allowlist

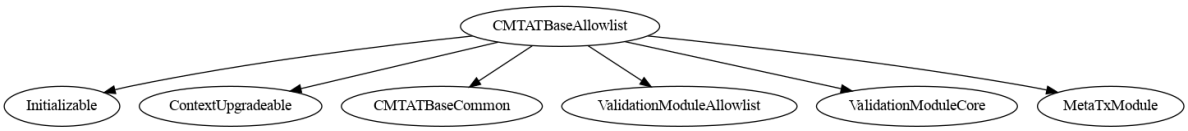
- CMTAT Standalone Allowlist



- 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 v0.1.0 (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

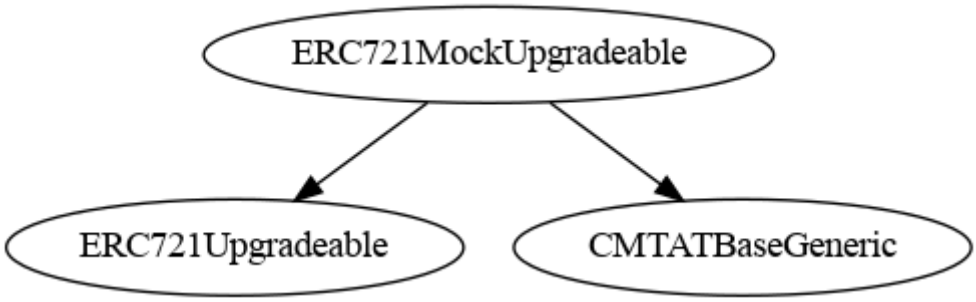
Further reading: [Taurus - Making CMTAT Tokenization More Scalable and Cost-Effective with Proxy and Factory Contracts] (version used CMTAT v2.5.1)

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

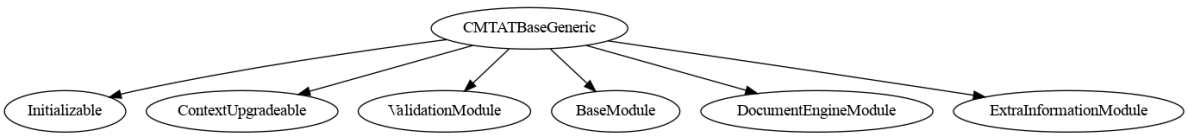
Deployment version using another type of tokens as ERC-20 (e.g ERC-721) or with a different logic (e.g [ZamaFHE - EncryptedERC20]) can be build by using the base contract `CMTATBaseGeneric`. This base contract inherits from several non-ERC-20 modules

Currently, there is no available version but a mock contract which implements ERC-721 with `CMTATBaseGeneric` is available in the mock directory: [contracts/mocks/EC721MockUpgradeable]

- ERC721MockUpgradeable



- CMTATBaseGeneric



Documentation

Here a summary of the main documents:

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

CMTA providers 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

Further reading

- 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 - Addressing the Privacy and Compliance Challenge in Public Blockchain Token Transactions (Aztec)

Security

Vulnerability disclosure

Please see SECURITY.md.

Module

See the code in modules/security.

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

Audit

The contracts have been audited by ABDKConsulting, a globally recognized firm specialized in smart contracts security.

First audit - September 2021

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

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.

Tools

Aderyn

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

Slither

You will find the report produced by Slither in

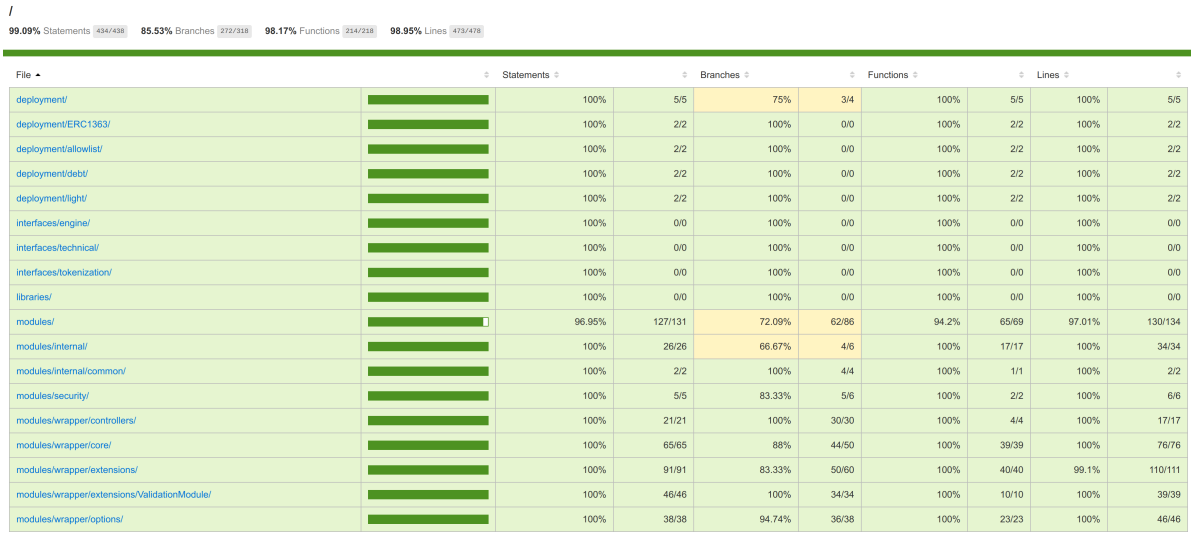
Version	File
v3.0.0	v3.0.0-slither-report.md
v2.5.0	v2.5.0-slither-report.md
v2.3.0	v2.3.0-slither-report.md

Mythril

Version	File
v2.5.0	mythril-report-standalone.md mythril-report-proxy.md

Test

A code coverage is available in `doc/test/coverage/index.html`


/

99.09% Statements 434/438 **85.53%** Branches 272/318 **98.17%** Functions 214/218 **98.95%** Lines 473/478

File		Statements		Branches		Functions		Lines	
deployment/		100%	5/5	75%	3/4	100%	5/5	100%	5/5
deployment/ERC1363/		100%	2/2	100%	0/0	100%	2/2	100%	2/2
deployment/allowlist/		100%	2/2	100%	0/0	100%	2/2	100%	2/2
deployment/debt/		100%	2/2	100%	0/0	100%	2/2	100%	2/2
deployment/light/		100%	2/2	100%	0/0	100%	2/2	100%	2/2
interfaces/engine/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
interfaces/technical/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
interfaces/tokenization/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
libraries/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
modules/		96.95%	127/131	72.09%	62/86	94.2%	65/69	97.01%	130/134
modules/internal/		100%	26/26	66.67%	4/6	100%	17/17	100%	34/34
modules/internal/common/		100%	2/2	100%	4/4	100%	1/1	100%	2/2
modules/security/		100%	5/5	83.33%	5/6	100%	2/2	100%	6/6
modules/wrapper/controllers/		100%	21/21	100%	30/30	100%	4/4	100%	17/17
modules/wrapper/core/		100%	65/65	88%	44/50	100%	39/39	100%	76/76
modules/wrapper/extensions/		100%	91/91	83.33%	50/60	100%	40/40	99.1%	110/111
modules/wrapper/extensions/ValidationModule/		100%	46/46	100%	34/34	100%	10/10	100%	39/39
modules/wrapper/options/		100%	38/38	94.74%	36/38	100%	23/23	100%	46/46

Remarks

As with any token contract, access to the owner key must be adequately restricted.
Likewise, access to the proxy contract must be restricted and seggregated from the token contract.

Other implementations

Two versions are available for the blockchain Tezos

- CMTAT FA2 Official version written in SmartPy
- @ligo/cmtat Unofficial version written in Ligo
    - See also Tokenization of securities on Tezos by Frank Hillard

A specific version is available for Aztec

- Aztec Private CMTAT
    - See also Taurus - Addressing the Privacy and Compliance Challenge in Public Blockchain Token Transactions

Configuration & toolchain

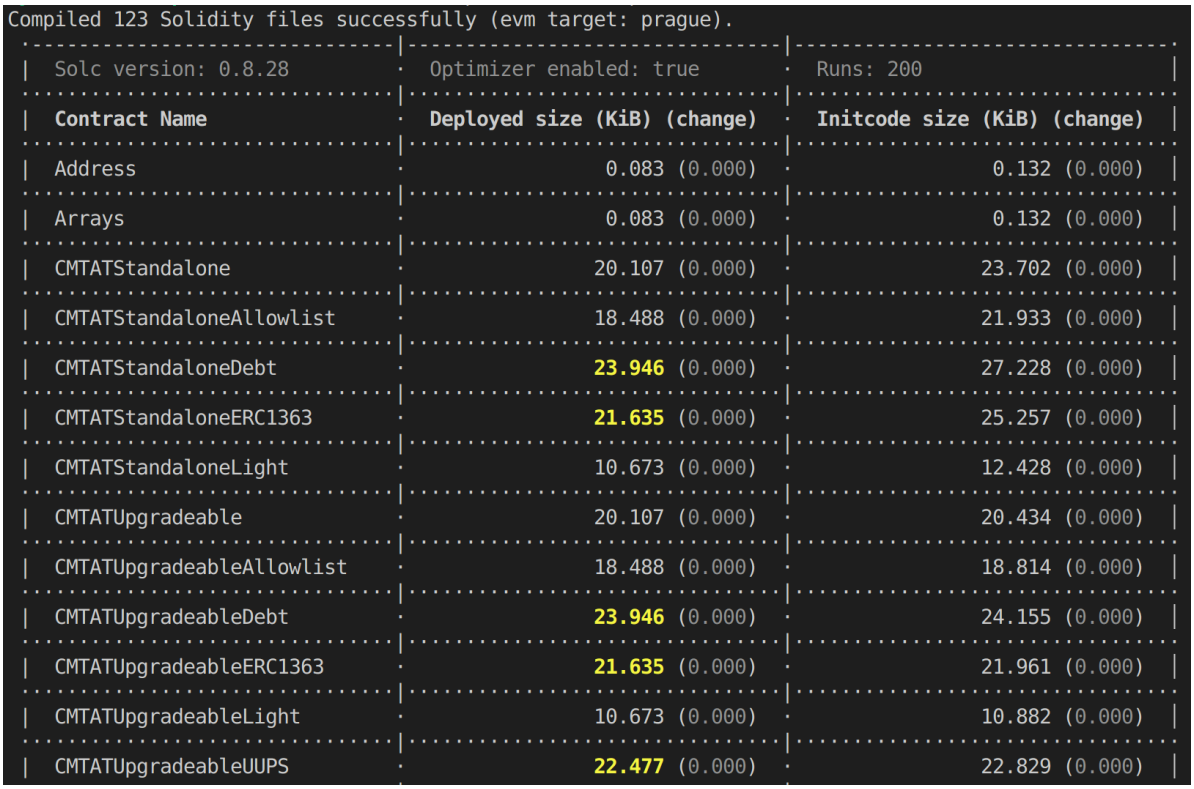
The project is built with Hardhat and uses OpenZeppelin

More information in doc/USAGE.md

- hardhat.config.js
    - Solidity 0.8.28
    - EVM version: Prague (Pectra upgrade)
    - Optimizer: true, 200 runs
- Package.json
    - OpenZeppelin Contracts (Node.js module): v5.3.0
    - OpenZeppelin Contracts Upgradeable (Node.js module): v5.3.0

Contract size

```
npm run-script size
```



```
Compiled 123 Solidity files successfully (evm target: prague).
·------------------------------|------------------------------|------------------------------·
|  Solc version: 0.8.28        ·  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)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandalone             ·               20.107 (0.000)  ·               23.702 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneAllowlist    ·               18.488 (0.000)  ·               21.933 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneDebt         ·               23.946 (0.000)  ·               27.228 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneERC1363      ·               21.635 (0.000)  ·               25.257 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATStandaloneLight        ·               10.673 (0.000)  ·               12.428 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeable            ·               20.107 (0.000)  ·               20.434 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableAllowlist   ·               18.488 (0.000)  ·               18.814 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableDebt        ·               23.946 (0.000)  ·               24.155 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableERC1363     ·               21.635 (0.000)  ·               21.961 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableLight       ·               10.673 (0.000)  ·               10.882 (0.000)  |
·------------------------------|------------------------------|------------------------------·
|  CMTATUpgradeableUUPS        ·               22.477 (0.000)  ·               22.829 (0.000)  |
```

Intellectual property