Digital Asset Rules, Set 1

APIs for interacting with Tezos STO token smart contracts

Version 0.3 (11/17/2020)

Authors:
Dominik Spicher <dominik.spicher@inacta.ch>
Roger Darin <roger.darin@inacta.ch>
Kevin Wespi <kevin.wespi@inacta.ch>
Daniel Gretzke <daniel.gretzke@inacta.ch>

Change History

- Version 0.1 (09/30/2020): Initial version
- Version 0.2 (10/27/2020): Rework of whitelisting APIs: consolidated six add/remove whitelist calls into three update calls, adjusted permissions
- Version 0.2.1 (11/04/2020): Fix title typos for APIs 8 and 9
- Version 0.2.2 (11/17/2020): Adjusted the API for document storage
- Version 0.3 (11/17/2020): Consolidated freeze and unfreeze into one function

1 Introduction

In the blockchain space in general, and on the Tezos blockchain in particular, there is a lot of interest in and momentum behind STOs, where a real-world asset is tokenized. The nature of these tokenized security offerings frequently requires smart contract functions that go beyond simple transfer functionality. For example, a designated owner address may be allowed to reissue tokens or whitelist a particular address before it may receive tokens.

Where banks and other financial institutions are involved, private-key handling is frequently outsourced to a third-party storage provider. The above-mentioned on-chain functionality is then accessed through APIs provided by the storage provider. Without proper standardization, this has the potential to lead to extensive fragmentation in terms of the provided APIs, and limits the possibility for an STO platform to support multiple storage providers.

Often, standardization is done on the smart contract level, for example via the well-known ERC specifications. From the above perspective, however, this has multiple drawbacks: First, there is ample reason for a smart contract standard to offer a larger feature set than might be warranted for the storage provider to support based on their customer demand. For example, even the relatively simple ERC-20 standard supports transfer on behalf of others which is often not needed. Also, the interests of a multitude of other stakeholders often results in very general and flexible APIs whose one-to-one implementation is prohibitive for storage providers to support from both a cost and security perspective.

Therefore, the goal of this document is to describe a minimal set of APIs as they are foreseen to be consumed by an STO platform. Without trying to anticipate the exact smart contract API, we believe this is the functionality that any well-received STO smart contract is bound to support. Note in particular that this includes the possibility that some of the functions described here end up being delegated to other smart contracts attached to the token smart contract.

This absolves interested integration parties from awaiting the results of lengthy standardization processes. We hope this document can help in driving discussions forward and in helping to ensure a vibrant STO landscape on the Tezos blockchain.

2 Preliminary Remarks

The next chapter will contain the description of the APIs. Here, we include some brief remarks that will help provide rationale and perspective for them and point out some limitations.

2.1 FA2 as a token basis

As mentioned in the introduction, the API standardization suggested here strives to stay independent of the underlying smart contract API. Nevertheless, we recognize that the FA2 standard has emerged as the clear leader in the Tezos token landscape. Where applicable,

therefore, we also briefly discuss how the proposed APIs map onto the versions provided by FA2.

One beneficial consequence of assuming an FA2-like token is the ability to assume multi-asset support. For the STO use-case, this can significantly simplify administration when multiple similar tokens can be subject to the same ownership and administration rules.

For reasons detailed in the introduction, we don't foresee full FA2 support to be necessary. Indeed, this document only makes use of the "transfer" FA2 endpoint.

2.2 Write operations only

This document focuses on smart contract interactions that change the underlying contract state. Read-only functions may also be desired, but they generally pose fewer integration hurdles. Also, in the case of off-chain use, they can be obtained by querying the node instead.

2.3 Token ownership

Some administrative functions listed below require access control. For simplicity, we assume that this is provided by a fixed contract owner address, specified at the time of contract creation. Ownership is shared among all tokens within a particular multi-asset contract.

2.4 Limitations

This document is not a specification. Differing circumstances and constraints among different storage providers preclude identical implementations. Given that, we still believe it is helpful to strive for a common feature set and to drive alignment as far as circumstances permit.

A particularly hard to foresee tradeoff involves the level of batch-processing provided by APIs. For example, the FA2 transfer function allows for batch-processing on multiple levels to save transaction costs. However, for security and user experience reasons, it is often advantageous to restrict the level of batch-processing. For simplicity therefore, this document presents all APIs in their singular form, while recognizing that more nested versions thereof will result in the end.

3 API descriptions

In the following, we describe APIs that STO platforms are likely to rely upon. They are grouped as follows:

1. Token transfer API
2. Security token APIs
3. Governance APIs
4. Document hash API

Each section will also give a brief description of the use-case involved. As mentioned, we assume multi-asset support and therefore include a `tokenId` argument to identify the asset where needed.

3.1 Token transfer API

We only consider token transfers originating from the sender address, i.e. no approved transfers on behalf of others.

API 1: Token transfer
`transfer(tokenId, to, amount)`
Callable by everyone.
Arguments:

- `tokenId` (number): identifies the token
- `to` (address): receiving address
- `amount` (number): number of tokens to transfer

Transfers tokens from the address where the transaction originated to the receiving address.

3.2 Security token APIs

This section contains administrative functions that are often part of security token standards.

API 2: Token issuance
`issue(tokenId, amount)`
Callable by the contract owner.
Arguments:

- `tokenId` (number): the token to be issued
- `amount` (number): number of tokens to be issued

Issues tokens to the owner address.

API 3: Token redemption (also known as “burning”)
`redeem(tokenId, amount)`
Callable by the contract owner.
Arguments:

- `tokenId` (number): the token to be redempted
- `amount` (number): number of tokens to be redeemed

Redeems tokens from the owner address.

API 4: Forced transfer
`reassign(from)`
Callable by the contract owner.
Arguments:

- `from` (address): the address from which tokens should be force-transferred

Assigns all tokens of a particular address to the token owner. Note the absence of the `tokenId` argument: All tokens are reassigned.

API 5: Freeze or unfreeze token transfers (also known as “pause”)

`setPause(bool pause)`

Callable by the contract owner.

Based on the parameter, freezes or unfreezes all token balances, i.e. no transfers can be performed anymore. The owner can still issue, redeem and reassign tokens while the contract is paused. Note the absence of the `tokenId` argument: Transfers are paused for all tokens.

3.3 Governance APIs

Security tokens often require some control over the investor universe. Whereas a multitude of possible on-chain governance mechanisms is possible, this document suggests the support for the most basic form of a whitelisting control: For every transfer, both the sender and the receiver need to be contained in a whitelist. The exception is the owner address, which we assume the implementation always treats as whitelisted.

For security tokens, it is likely that whitelisting governance functions will be required to be delegatable to non-owners of a specific token contract (e.g. a bank that whitelists its clients). Thus, we propose an administration scheme that knows two roles:

1. Whitelist admins, the addition and removal of which is restricted to the owner
2. Whitelisters, the addition and removal of which is restricted to whitelist admins

The whitelisting and de-whitelisting of addresses itself is then performed by whitelisters.

The above scheme results in the following APIs.

The following modification parameter is used by all whitelist APIs:

`modify_parameter: [{add: address}, {remove: address}, …]`

The parameter is a list of instuctions to either `add` or `remove` an entry.

API 6: Modify whitelist admins

`modifyWhitelistAdmins(modify_parameter)`

Callable by the owner of the contract.

Arguments:

- `modify_parameter`: addresses to add or remove from the whitelist admin list

Adds and/or removes a set of addresses to the set of whitelist admins.

API 7: Modify whitelisters

`modifyWhitelisters(modify_parameter)`

Callable by a whitelist admin.

Arguments:

- `modify_parameter`: addresses to add or remove from the whitelister list

Adds and/or removes a set of addresses to the set of whitelisters.

API 8: Modify whitelisted addresses

`modifyWhitelist(modify_parameter)`

Callable by a whitelister.

Arguments:

- `modify_parameter`: addresses to add or remove from the whitelist

Adds and/or removes a set of addresses from the whitelist.

3.4 Document hash API

STOs frequently come with a prospectus that is distributed to issuers. To be able to link the token smart contract with such artifacts, the contract supports the hash commitment to a document identified by its name. This function can be called multiple times, and implementations are expected to collect historical commitments in a suitable datastructure.

API 9: Add document

`addDocument(documentName, hash)`

Callable by the owner.

Arguments:

1. `documentName` (string): the name of the document
2. `hash` (string): the hash of the document

Adds a hash for a particular document to the list of historical hashes.