Summary

There is a need for a minimal abstract ledger that can be used as a component in applications requiring some notion of fungible asset (or "token"). The space of all possible contracts that require fungibility is vast, so this standard is defined in the most general possible way.

This document describes an interface for a ledger that maps identities to balances. This standard targets library, client tools and contract developers who want to interact with fungible assets. More concrete standards are expected to extend this abstract ledger and preserve compatibility.

Scope

This abstract ledger intentionally lacks certain features often desired when implementing a fungible asset contract. It neither specifies how other contracts should operate with users' funds, nor provides any specific mechanisms for contracts to record incoming transactions. Other important features like fallback addresses and monetary supply management are left to developers or extensions of this standard as well.

Abstract Ledger Interface

This interface relies on multiple entrypoints feature. According to it, parameter of any contract implementing the interface should be a tree of `Or`s, some of its leaves should correspond to interface methods.

For FA1, parameter should contain the following leaves:

1. `(address :from, (address :to, nat :value)) %transfer`
2. `view (address :owner) nat %getBalance`
3. `view unit nat %getTotalSupply`

See also syntax explanation and Michelson Contract Interfaces and Conventions Document.

Errors

Failures of this contract are represented as `(string, d)` pairs, the first element of which is an identifier of an error and the second element keeps details of this error, or `unit` if no details required.

For example, attempt to withdraw `5` tokens when only `3` is present will result in the following error: `("NotEnoughBalance", (5, 3))`

We will express the meaning of particular error details in their type for the sake of convenience. For instance, we say that `NotEnoughBalance` error carries a value of `(nat :required, nat :present)` type. This does not mean that value passed to `FAILWITH` call should not necessarily be annotated because that would not affect the produced error anyway, rather we use annotations just as a designation.

Entrypoints

transfer

This entrypoint credits the account of the address passed in the `:to` parameter, while debiting the account matching the `:from` address. In case the `:from` address has insufficient funds, the transaction MUST fail and no state should be mutated. Total supply MUST NOT be modified upon `transfer`.

Additional authorization checks are to be defined by a concrete implementation or specified in extension standards.

In case of insufficient balance, this entrypoint must fail with the following error:

- `NotEnoughBalance` - insufficient funds on the sender account to perform a given transfer. The error must contain a `(nat :required, nat :present)` pair, where `required` is the requested amount of tokens, `present` is the available amount.

getTotalSupply

This view returns the sum of all participants' balances.

getBalance

This view returns balance of the given address, or zero if no such address is registered.