Table Of Contents

Summary

TZIP-012 proposes a standard for a unified token contract interface, supporting a wide range of token types and implementations. This document provides an overview of the interface, token transfer semantics, and metadata.

Abstract

Many considerations weigh on the implementer of a token contract. Tokens might be fungible or non-fungible. A variety of transfer permission policies can be used to define how many tokens can be transferred, who can perform a transfer, and who can receive tokens. A token contract can be designed to support a single token type (e.g. ERC-20 or ERC-721) or multiple token types (e.g. ERC-1155) to optimize batch transfers and atomic swaps of the tokens.

The FA2 standard aims to provide significant expressivity to contract developers to create new types of tokens while maintaining a common interface standard for wallet integrators and external developers.

A particular FA2 implementation may support either a single token type per contract or multiple tokens per contract, including hybrid implementations where multiple token kinds (fungible, non-fungible, non-transferable etc) can coexist (e.g. in a fractionalized NFT contract).

This document also specifies metadata at the token and contract level based on TZIP-016. It also defines the set of standard errors and error mnemonics to be used when implementing FA2.

General

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

- Token type is uniquely identified on the chain by a pair composed of the token contract address and token ID, a natural number (`nat`). If the underlying contract implementation supports only a single token type (e.g. ERC-20-like contract), the token ID MUST be `0n`. In the case when multiple token types are supported within the same FA2 token contract (e. g. ERC-1155-like contract), the contract is fully responsible for assigning and managing token IDs.
- The FA2 batch entrypoints accept a list (batch) of parameters describing a single operation or a query. The batch MUST NOT be reordered or deduplicated and MUST be processed in the same order it is received.
- Empty batch is a valid input and MUST be processed as a non-empty one. For example, an empty transfer batch will not affect token balances, but applicable transfer core behavior and permission policy MUST be applied. Invocation of the `balance_of` entrypoint with an empty batch input MUST result in a call to a callback contract with an empty response batch.
- If the underlying contract implementation supports only a single token type, the batch may contain zero or multiple entries where token ID is a fixed `0n` value. Likewise, if multiple token types are supported, the batch may contain zero or more entries and there may be duplicate token IDs.

Interface Specification

Token contract implementing the FA2 standard MUST have the following **Michelson** entrypoints: `transfer`, `balance_of`, `update_operators`

Entrypoint Semantics

`transfer`

```
(list %transfer
  (pair
    (address %from_)
    (list %txs
      (pair
        (address %to_)
        (pair
          (nat %token_id)
          (nat %amount)
        )
      )
    )
  )
)
```

Each transfer in the batch is specified between one source (`from_`) address and a list of destinations. Each `transfer_destination` specifies token type and the amount to be transferred from the source address to the destination (`to_`) address.

FA2 does NOT specify an interface for mint and burn operations; however, if an FA2 token contract implements mint and burn operations, it SHOULD, when possible, enforce the same logic (core transfer behavior and transfer permission logic) applied to the token transfer operation. Mint and burn can be considered special cases of the transfer. Although, it is possible that mint and burn have more or less restrictive rules than the regular transfer. For instance, mint and burn operations may be invoked by a special privileged administrative address only. In this case, regular operator restrictions may not be applicable.

Core Transfer Behavior

FA2 token contracts MUST always implement this behavior.

- Every transfer operation MUST happen atomically and in order. If at least one transfer in the batch cannot be completed, the whole transaction MUST fail, all token transfers MUST be reverted, and token balances MUST remain unchanged.
- Each transfer in the batch MUST decrement token balance of the source (`from_`) address by the amount of the transfer and increment token balance of the destination (`to_`) address by the amount of the transfer.
- If the transfer amount exceeds current token balance of the source address, the whole transfer operation MUST fail with the error mnemonic `"FA2_INSUFFICIENT_BALANCE"`.
- If the token owner does not hold any tokens of type `token_id`, the owner's balance is interpreted as zero. No token owner can have a negative balance.
- The transfer MUST update token balances exactly as the operation parameters specify it. Transfer operations MUST NOT try to adjust transfer amounts or try to add/remove additional transfers like transaction fees.
- Transfers of zero amount MUST be treated as normal transfers.
- Transfers with the same address (`from_` equals `to_`) MUST be treated as normal transfers.
- If one of the specified `token_id`s is not defined within the FA2 contract, the entrypoint MUST fail with the error mnemonic `"FA2_TOKEN_UNDEFINED"`.
- Transfer implementations MUST apply transfer permission policy logic (either default transfer permission policy or customized one). If permission logic rejects a transfer, the whole operation MUST fail.
- Core transfer behavior MAY be extended. If additional constraints on tokens transfer are required, FA2 token contract implementation MAY invoke additional permission policies. If the additional permission fails, the whole transfer operation MUST fail with a custom error mnemonic.

Default Transfer Permission Policy

- Token owner address MUST be able to perform a transfer of its own tokens (e. g. `SENDER` equals to `from_` parameter in the `transfer`).
- An operator (a Tezos address that performs token transfer operation on behalf of the owner) MUST be permitted to manage the specified owner's tokens before it invokes a transfer transaction (see

update_operators).

- If the address that invokes a transfer operation is neither a token owner nor one of the permitted operators, the transaction MUST fail with the error mnemonic "FA2_NOT_OPERATOR". If at least one of the transfers in the batch is not permitted, the whole transaction MUST fail.

balance_of

```
(pair %balance_of
  (list %requests
    (pair
      (address %owner)
      (nat %token_id)
    )
  )
  (contract %callback
    (list
      (pair
        (pair %request
          (address %owner)
          (nat %token_id)
        )
        (nat %balance)
      )
    )
  )
)
```

Gets the balance of multiple account/token pairs. Accepts a list of balance_of_requests and a callback contract callback which accepts a list of balance_of_response records.

- There may be duplicate balance_of_request's, in which case they should not be deduplicated nor reordered.
- If the account does not hold any tokens, the account balance is interpreted as zero.
- If one of the specified token_ids is not defined within the FA2 contract, the entrypoint MUST fail with the error mnemonic "FA2_TOKEN_UNDEFINED".

Notice: The balance_of entrypoint implements a *continuation-passing style (CPS) view entrypoint* pattern that invokes the other callback contract with the requested data. This pattern, when not used carefully, could expose the callback contract to an inconsistent state and/or manipulatable outcome (see view patterns). The balance_of entrypoint should be used on the chain with extreme caution.

Operators

Operator is a Tezos address that originates token transfer operation on behalf of the owner.

Owner is a Tezos address which can hold tokens.

An operator, other than the owner, MUST be approved to manage specific tokens held by the owner to transfer them from the owner account.

FA2 interface specifies an entrypoint to update operators. Operators are permitted per specific token owner and token ID (token type). Once permitted, an operator can transfer tokens of that type belonging to the owner.

`update_operators`

```
(list %update_operators
  (or
    (pair %add_operator
      (address %owner)
      (pair
        (address %operator)
        (nat %token_id)
      )
    )
    (pair %remove_operator
      (address %owner)
      (pair
        (address %operator)
        (nat %token_id)
      )
    )
  )
)
```

Add or Remove token operators for the specified token owners and token IDs.

- The entrypoint accepts a list of `update_operator` commands. If two different commands in the list add and remove an operator for the same token owner and token ID, the last command in the list MUST take effect.
- It is possible to update operators for a token owner that does not hold any token balances yet.
- Operator relation is not transitive. If C is an operator of B and if B is an operator of A, C cannot transfer tokens that are owned by A, on behalf of B.

The standard does not specify who is permitted to update operators on behalf of the token owner. Depending on the business use case, the particular implementation of the FA2 contract MAY limit operator updates to a token owner (`owner == SENDER`) or be limited to an administrator.

Token Metadata

Token metadata is intended for off-chain, user-facing contexts (e.g. wallets, explorers, marketplaces). An earlier (superseded) specification of TZIP-012 token metadata is contained in the *Legacy Interface* section of the Legacy FA2 document.

Token-Metadata Values

Token-specific metadata is stored/presented as a Michelson value of type `(map string bytes)`. A few of the keys are reserved and predefined by TZIP-012:

- `""` (empty-string): should correspond to a TZIP-016 URI which points to a JSON representation of the token metadata.
- `"name"`: should be a UTF-8 string giving a “display name” to the token.
- `"symbol"`: should be a UTF-8 string for the short identifier of the token (e.g. XTZ, EUR, …).
- `"decimals"`: should be an integer (converted to a UTF-8 string in decimal) which defines the position of the decimal point in token balances for display purposes.

In the case of a TZIP-016 URI pointing to a JSON blob, the JSON preserves the same 3 reserved non-empty fields:

```
{ "symbol": <string>, "name": <string>, "decimals": <number>, ... }
```

Providing a value for `"decimals"` is required for all token types. `"name”` and `"symbol"` are not required but it is highly recommended for most tokens to provide the values either in the map or the JSON found via the TZIP-016 URI.

Other standards such as TZIP-021 describe token metadata schemas which reserve additional keys for different token types for greater compatibility across indexers, wallets, and tooling.

Token Metadata Storage & Access

A contract can use two methods to provide access to the token-metadata.

- **Basic**: Store the values in a big-map annotated `%token_metadata` of type `(big_map nat (pair (nat %token_id) (map %token_info string bytes)))`.

- **Custom**: Provide a `token_metadata` off-chain-view which takes as parameter the `nat` token-id and returns the `(pair (nat %token_id) (map %token_info string bytes))` value.

In both cases the “key” is the token-id (of type `nat`) and one MUST store or return a value of type `(pair nat (map string bytes))`: the token-id and the metadata defined above.

If both options are present, it is recommended to give precedence to the the off-chain-view (custom).

Contract Metadata (TZIP-016)

An FA2-compliant contract SHOULD provide contract-level metadata via TZIP-016:

- If a contract does not contain the TZIP-016 `%metadata` big-map, it must provide token-specific-metadata through the `%token_metadata` big-map method described above in Token Metadata.

The TZIP-016 contract metadata JSON structure is described below:

- The TZIP-016 `"interfaces"` field MUST be present
    - It should contain `"TZIP-012[-<version-info>]"`
        - `version-info` is an optional string extension, precising which version of this document is implemented by the contract (commit hash prefix, e.g. `6883675` or an RFC-3339 date, e.g. `2020-10-23`).

- The TZIP-016 `"views"` field is optional, a few optional off-chain-views are specifed below, see section Off-chain-views.
- A TZIP-012-specific field `"permissions"` is defined in Exposing Permissions Descriptor, and it is optional, but recommended if it differs from the default value.

Example

A single-NFT FA2 token can be augmented with the following JSON:

```
{
  "description": "This is my NFT",
  "interfaces": ["TZIP-012-2020-11-17"],
  "views": [
    { "name": "get_balance",
      "description": "This is the `get_balance` view required by TZIP-012.",
      "implementations": [
          { "michelsonStorageView": {
              "parameter": {
                  "prim": "pair",
                  "args": [{"prim": "address", "annots": ["%owner"]},
                           {"prim": "nat", "annots": ["token_id"]}]},
              "returnType": {"prim": "nat"},
              "code": [
                  {"prim": "TODO"}]}}]}]
}
```

Off-Chain-Views

Within its TZIP-016 metadata, an FA2 contract does not have to provide any off-chain-view but can provide 5 optional views: `get_balance`, `total_supply`, `all_tokens`, `is_operator`, and `token_metadata`. If present, all of these SHOULD be implemented, at least, as *"Michelson Storage Views"* and have the following types (Michelson annotations are optional) and semantics:

- `get_balance` has `(pair (address %owner) (nat %token_id))` as parameter-type, and `nat` as return-type; it must return the balance corresponding to the owner/token pair.
- `total_supply` has type `(nat %token_id) → (nat %supply)` and should return to total number of tokens for the given token-id if known or fail if not.
- `all_tokens` has no parameter and returns the list of all the token IDs, `(list nat)`, known to the contract.
- `is_operator` has type `(pair (address %owner) (pair (address %operator) (nat %token_id))) → bool` and should return whether `%operator` is allowed to transfer `%token_id` tokens owned by `owner`.
- `token_metadata` is one of the 2 ways of providing token-specific metadata, it is defined in section Token Metadata and is not optional if the contract does not have a `%token_metadata` big-map.

Token balance updates

FA2 contracts usually have non-standard (other than `transfer`) entrypoints that alter token balances. In addition to that, there may be an initial distribution of tokens at the origination of the contract. All that makes impossible for a third-party that is working with the FA2 contract in a generic way (not knowing the implementation details) to do proper token balance accounting.

One can make token balances indexable by storing them in a standardized way. Depending on the case, one of the following options should be used.

Single asset contract

```
big_map %ledger address nat
```

where key is the owner's address and value is the amount of tokens owned.

Multi asset contract

```
big_map %ledger (pair address nat) nat
```

where key is the pair [owner's address, token ID] and value is the amount of tokens owned.

NFT asset contract

```
big_map %ledger nat address
```

where key is the token ID and value is owner's address.

FA2 Transfer Permission Policies and Configuration

Most token standards specify logic such as who can perform a transfer, the amount of a transfer, and who can receive tokens. This standard calls such logic *transfer permission policy* and defines a framework to compose such permission policies from the standard permission behaviors.

FA2 allows the contract developer to choose and customize from a variety of permissions behaviors, easily enabling non-transferrable tokens or centrally-administrated tokens without operators. The particular implementation may be static (the permissions configuration cannot be changed after the contract is deployed) or dynamic (the FA2 contract may be upgradable and allow to change the permissions configuration). However, the FA2 token contract MUST expose consistent and non-self-contradictory permissions configuration (unlike ERC-777 that exposes two flavors of the transfer at the same time).

A full treatment of FA2 Permission Policies and configuration can be found in a dedicated Permission Policy document.

Error Handling

This specification defines the set of standard errors to make it easier to integrate FA2 contracts with wallets, DApps and other generic software, and enable localization of user-visible error messages.

Each error code is a short abbreviated string mnemonic. An FA2 contract client (like another contract or a wallet) could use on-the-chain or off-the-chain registry to map the error code mnemonic to a user-readable, localized message. A particular implementation of the FA2 contract MAY extend the standard set of errors with custom mnemonics for additional constraints.

Standard error mnemonics:

Error mnemonic	Description
`"FA2_TOKEN_UNDEFINED"`	One of the specified `token_id`s is not defined within the FA2 contract
`"FA2_INSUFFICIENT_BALANCE"`	A token owner does not have sufficient balance to transfer tokens from owner's account
`"FA2_TX_DENIED"`	A transfer failed because of `operator_transfer_policy == No_transfer`
`"FA2_NOT_OWNER"`	A transfer failed because `operator_transfer_policy == Owner_transfer` and it is invoked not by the token owner
`"FA2_NOT_OPERATOR"`	A transfer failed because `operator_transfer_policy == Owner_or_operator_transfer` and it is invoked neither by the token owner nor a permitted operator
`"FA2_OPERATORS_UNSUPPORTED"`	`update_operators` entrypoint is invoked and `operator_transfer_policy` is `No_transfer` or `Owner_transfer`
`"FA2_RECEIVER_HOOK_FAILED"`	The receiver hook failed. This error MUST be raised by the hook implementation
`"FA2_SENDER_HOOK_FAILED"`	The sender failed. This error MUST be raised by the hook implementation
`"FA2_RECEIVER_HOOK_UNDEFINED"`	Receiver hook is required by the permission behavior, but is not implemented by a receiver contract
`"FA2_SENDER_HOOK_UNDEFINED"`	Sender hook is required by the permission behavior, but is not implemented by a sender contract

If more than one error conditions are met, the entrypoint MAY fail with any applicable error.

When an error occurs, any FA2 contract entrypoint MUST fail with one of the following types:

1. `string` value which represents an error code mnemonic.
2. a Michelson `pair`, where the first element is a `string` representing error code mnemonic and the second element is a custom error data.

Some FA2 implementations MAY introduce their custom errors that MUST follow the same pattern as standard ones: define custom error mnemonics and fail with one of the error types defined above.

Implementing Different Token Types With FA2

The FA2 interface is designed to support a wide range of token types and implementations. The FA2 implementation guide provide examples of how different types of the FA2 contracts MAY be implemented and what are the expected properties of such an implementation.

Future Directions

Several **backwards-compatible** TZIP-012 upgrades are likely depending on proposed Tezos protocol amendments:

- Views and Events
  - Extending TZIP-012 to include on-chain views (and removal of `balance_of`). Reduce complexity of off-chain views and events.
- Tickets
  - Extending TZIP-012 to allow wrapping of FA2 tokens as tickets and marking `update_operators` as optional.

Copyright