## Summary

An extension of the Contract Metadata standard specifying new kinds of Off-chain Views which are intended to derive an additional receipt from the operation's content and result without extra context queries.

## Abstract

Off-chain events is an external contract event logging system proposed as a workaround until a native one is available in Tezos VM. The main driver for that is a need for customizable indexing of contract calls and metadata. The idea is to let contract developers to write the custom logic in a way that:

- It does not add extra gas/storage costs;
- It can be applied retrospetively, meaning that one can introduce (or alter) custom handlers AFTER contract is originated;
- It can be adopted by anyone, i.e. this information is not restricted or tied to a specific implementation.

Off-chain events are very similar to off-chain views defined in the TZIP-16 standard, though have several differences that will be covered in the next section.
In a nutshell, off-chain event is a Michelson script that accepts operation content (e.g. parameters) or result (storage, big_map_diff) as **parameter**, empty value as **storage**, and returns an empty list of **operations** and the new **storage** containing event logs.

TZIP-20 defines:

- An extensibe JSON format for describing off-chain events;
- Several event kinds to handle various parts of the operation content and result;
- An approach to notify indexers when token metadata changes or a non-standard token balance update occurs.

## Differences from off-chain views

Off-chain events are:

- Always pure, i.e. same inputs will result in same values;
- Not allowing *Big_maps* and context dependant instructions `CONTRACT`, `SELF`, `SELF_ADDRESS`, `BALANCE`;
- Mocking several instructions with values deduced from operation parameters `SOURCE`, `SENDER`, `AMOUNT`, `CHAIN_ID`, `NOW`, `LEVEL`.

## JSON format

Events are embedded into the TZIP-16 JSON file: list of `"events"` is available at the root level in case `"TZIP-20"` is present in the `"interfaces"` array.

Example:

```
{
  "interfaces": [ "TZIP-20" ],
  "events": [
     //... see below ...
  ]
  // ... other TZIP-16 fields ...
}
```

An off-chain event object has at least 2 fields:

- `"name"`; the canonical name of the event defining the return value type (Michelson storage type).
- `"description"` *(optional)*: a human readable description of the behavior of the event.
- `"implementations"`: a list of implementation objects: usable definitions of the events. Each implementation is a one-field object where the field name discriminates between various kinds of events. Below, this standard defines 3 of those kinds, `"michelsonParameterEvent"`, `"michelsonInitialStorageEvent"`, and `"michelsonExtendedStorageEvent"`.

Example:

```
{
  "name": "singleAssetBalanceUpdates",
  "description": "Get token balance updates as map { account => delta } based on tx params (optionally reduced)",
  "implementations": [
     { "michelsonParameterEvent" : { /* ... see below ... */ } },
     { "michelsonInitialStorageEvent" : { /* ... see below ... */ } },
     { "michelsonExtendedStorageEvent" : { /* ... see below ... */ } },
     // ... potential extensions ...
  ]
}
```

### Michelson parameter event

The `"michelsonParameterEvent"` field is a JSON object describing a sequence of Michelson instructions to run on a pair formed by a transaction parameter and an empty storage of the type implied by the event name in order to leave the execution stack with the event logs.
For this object we define 4 fields (all required):

- `"parameter"`: either `parameter` type of the contract, or a "reduced" type — a subset of contract entrypoints (requires data normalization).
- `"returnType"`: Michelson type of the value left on the stack (must have an unambiguous initial value).
- `"code"`: Michelson code expression implementing the event (a sequence of instructions).
- `"entrypoints"`: list of entrypoints that trigger this event implementation.

The first three "Michelson" fields have the same format, they are JSON values obeying the Michelson JSON format of the Tezos protocol (sometimes referred to as "Micheline" encoding).

Example:

```
{
    "parameter": {"prim": "or", "args": [
        {"prim": "pair", "annots": ["%mint"], "args": [{"prim": "address"},
{"prim": "nat"}]},
        {"prim": "nat", "annots": ["%burn"]}
    ]},
    "returnType": {"prim": "map", "args": [{"prim": "address"}, {"prim": "int"}]},
    "code": [
        {"prim": "DUP"},
        {"prim": "CDR"},
        {"prim": "SWAP"},
        {"prim": "CAR"},
        {
            "prim": "IF_LEFT",
            "args": [
                [{"prim": "DUP"}, {"prim": "CDR"}, {"prim": "INT"}, {"prim":
"SOME"}, {"prim": "SWAP"}, {"prim": "CAR"}],
                [{"prim": "INT"}, {"prim": "NEG"}, {"prim": "SOME"}, {"prim":
"SENDER"}]
            ]
        },
        {"prim": "UPDATE"},
        {"prim": "NIL", "args": [{"prim": "operation"}]},
        {"prim": "PAIR"}
    ],
    "entrypoints": ["mint", "burn"]
}
```

### Michelson initial storage event

The `"michelsonInitialStorageEvent"` accepts initial contract storage at the origination as a parameter. For this object we define 3 fields (all required):

- `"parameter"`: equals to the contract `storage` type except for *Big_maps* that should be replaced by plain *Maps*, it allows you to iterate over lazily stored data (see data normalization).
- `"returnType"`: Michelson type of the event logs (i.e. for the value left on the stack).
- `"code"`: Michelson code expression implementing the event (a sequence of instructions).

**Example**

```
{
  "parameter": {
    "prim": "map", "annots": ["%ledger"], "args": [{"prim": "pair", "args":
[{"prim": "address"}, {"prim": "nat"}]}, {"prim": "nat"}]
  },
  "returnType": {
    "prim": "map", "args": [{"prim": "pair", "args": [{"prim": "address"},
{"prim": "nat"}]}, {"prim": "int"}]
  },
  "code": [
    {"prim": "CAR"},
    {"prim": "MAP", "args": [[{"prim": "CDR"}, {"prim": "INT"}]]},
    {"prim": "NIL", "args": [{"prim": "operation"}]}, {"prim": "PAIR"}
  ]
}
```

### Michelson extended storage event

The `"michelsonExtendedStorageEvent"` accepts storage from the operation result where all *Big_map* pointers are replaced with *Big_map_diffs* aggregated by *Big_map_id* — **extended storage** (see data normalization). For this object we define 4 fields (all required):

- `"parameter"`: equals to the contract `storage` type except for *Big_maps* that should be replaced by plain *Maps*, it allows you to iterate over lazily stored data and work with storage as a whole (see data normalization). Additionally *Big_map* values have to be wrapped with options in order not to lose information about removals.
- `"returnType"`: Michelson type of the event logs (i.e. for the value left on the stack).
- `"code"`: Michelson code expression implementing the event (a sequence of instructions).
- `"entrypoints"`: list of entrypoints that trigger this event implementation.

**Example**

```
{
  "parameter": {
    "prim": "map", "annots": ["%ledger"], "args": [{"prim": "pair", "args":
[{"prim": "address"}, {"prim": "nat"}]}, {"prim": "option", "args": [{"prim":
"nat"}]}]
  },
  "returnType": {
    "prim": "map", "args": [{"prim": "pair", "args": [{"prim": "address"},
{"prim": "nat"}]}, {"prim": "int"}]
  },
  "code": [
    {"prim": "CAR"},
    {"prim": "MAP", "args": [[{"prim": "CDR"}, {"prim": "IF_NONE", "args": [
      [{"prim": "PUSH", "args": [{"prim": "int"}, {"int": "0"}]}],
      [{"prim": "INT"}]
    ]}]]},
    {"prim": "NIL", "args": [{"prim": "operation"}]}, {"prim": "PAIR"}
  ],
  "entrypoints": ["mint"]
}
```

## Indexer events

This specification defines an approach to emitting token balance and metadata update events, which requires implementing off-chain events with standardized names and return types.

> Here and hereafter, we are based on the following statements about tokens:
>
> 1. Token represents an asset;

> 2. Single smart contract can contain multiple tokens;
> 3. Within a single contract a token is identified by a natural number.

NOTE that this specification doesn't linked to any particular token standard.

Read more in the indexer events documentation.

## Implementation guide

- For contract developers
- For indexer developers