



main ▾

bigmarket-dao / contracts

/ bigmarket-dao.clar

Go to file



mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

767bb51 · 3 weeks ago

Executable File · 86 lines (69 loc) · 2.72 KB

Code

Blame

Raw




```
1  ;; BigMarket DAO
2  ;; Synopsis:
3  ;; bigmarket-dao is the core of the dao framework.
4  ;; Description:
5  ;; Valid extensions must be registered here. The DAO is bootstrapped
6  ;; by calling construct with a bootstrap proposal.
7
8  (use-trait extension-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
9  (use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-trait)
10
11 (define-constant err-unauthorised (err u1000))
12 (define-constant err-already-executed (err u1001))
13 (define-constant err-invalid-extension (err u1002))
14
15 (define-data-var executive principal tx-sender)
16 (define-map executed-proposals principal uint)
17 (define-map extensions principal bool)
18
19 ;; --- Authorisation check
20
21 (define-private (is-self-or-extension)
22   (ok (asserts! (or (is-eq tx-sender (as-contract tx-sender)) (is-extension contract-ca
23   )
24
25 ;; --- Extensions
26
27 (define-read-only (is-extension (extension principal))
28   (default-to false (map-get? extensions extension)))
29 )
30
31 (define-public (set-extension (extension principal) (enabled bool))
32   (begin
33     (try! (is-self-or-extension))
34     (print {event: "extension", extension: extension, enabled: enabled})
35     (ok (map-set extensions extension enabled))
36   )
37 )
```

```

38
39 (define-private (set-extensions-iter (item {extension: principal, enabled: bool}))
40     (begin
41         (print {event: "extension", extension: (get extension item), enabled: (get er
42             (map-set extensions (get extension item) (get enabled item))
43         )
44     )
45
46 (define-public (set-extensions (extension-list (list 200 {extension: principal, enabled: bool
47     (begin
48         (try! (is-self-or-extension))
49         (ok (map set-extensions-iter extension-list))
50     )
51 )
52
53 ;; --- Proposals
54
55 (define-read-only (executed-at (proposal <proposal-trait>))
56     (map-get? executed-proposals (contract-of proposal))
57 )
58
59 (define-public (execute (proposal <proposal-trait>) (sender principal))
60     (begin
61         (try! (is-self-or-extension))
62         (asserts! (map-insert executed-proposals (contract-of proposal) stacks-block-
63             (print {event: "execute", proposal: proposal})
64             (as-contract (contract-call? proposal execute sender))
65         )
66     )
67
68 ;; --- Bootstrap
69
70 (define-public (construct (proposal <proposal-trait>))
71     (let ((sender tx-sender))
72         (asserts! (is-eq sender (var-get executive)) err-unauthorised)
73         (var-set executive (as-contract tx-sender))
74         (as-contract (execute proposal sender))
75     )
76 )
77
78 ;; --- Extension requests
79
80 (define-public (request-extension-callback (extension <extension-trait>) (memo (buff 34)))
81     (let ((sender tx-sender))
82         (asserts! (is-extension contract-caller) err-invalid-extension)
83         (asserts! (is-eq contract-caller (contract-of extension)) err-invalid-extensi
84         (as-contract (contract-call? extension callback sender memo))
85     )
86 )

```

 [main](#) [bigmarket-dao / contracts / traits](#)
[/ extension-trait.clar](#) 

 Go to file

...



mijoco-btc first commit

133764b · 3 months ago



Executable File · 5 lines (5 loc) · 94 Bytes

Code

Blame

Raw



```
1  (define-trait extension-trait
2      (
3          (callback (principal (buff 34)) (response bool uint))
4      )
5  )
```


main bigmarket-dao / contracts / traits / governance-token-trait.clar

Go to file

mijoco-btc Renamed contracts, updated heading comments, tweaked the custom major...

79c28af · 2 months ago

Executable File · 12 lines (12 loc) · 490 Bytes

Code Blame

Raw



```
1  (define-trait governance-token-trait
2      (
3          (bmg-get-balance (principal) (response uint uint))
4          (bmg-has-percentage-balance (principal uint) (response bool uint))
5          (bmg-transfer (uint principal principal) (response bool uint))
6          (bmg-lock (uint principal) (response bool uint))
7          (bmg-unlock (uint principal) (response bool uint))
8          (bmg-get-locked (principal) (response uint uint))
9          (bmg-mint (uint principal) (response bool uint))
10         (bmg-burn (uint principal) (response bool uint))
11     )
12 )
```


main **bigmarket-dao / contracts / traits**
/ prediction-market-trait.clar

Go to file



mijoco-btc Inclusion of reputation soul bound sft.

32188d4 · last week



6 lines (6 loc) · 169 Bytes

Code

Blame

Raw



```
1  (define-trait prediction-market-trait
2    (
3      (dispute-resolution (uint principal) (response bool uint))
4      (resolve-market-vote (uint uint) (response bool uint))
5    )
6  )
```




mijoco-btc

first commit

133764b · 3 months ago



Executable File · 5 lines (5 loc) · 82 Bytes

Code

Blame

Raw



```
1  (define-trait proposal-trait
2      (
3          (execute (principal) (response bool uint))
4      )
5  )
```




main ▾

[bigmarket-dao / contracts / traits](#)[/ sip013-semi-fungible-token-trait.clar](#)

Go to file

**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



27 lines (20 loc) · 961 Bytes


Code

Blame

Raw



```
1      (define-trait sip013-semi-fungible-token-trait
2          (
3              ;; Get a token type balance of the passed principal.
4              (get-balance (uint principal) (response uint uint))
5
6              ;; Get the total SFT balance of the passed principal.
7              (get-overall-balance (principal) (response uint uint))
8
9              ;; Get the current total supply of a token type.
10             (get-total-supply (uint) (response uint uint))
11
12             ;; Get the overall SFT supply.
13             (get-overall-supply () (response uint uint))
14
15             ;; Get the number of decimal places of a token type.
16             (get-decimals (uint) (response uint uint))
17
18             ;; Get an optional token URI that represents metadata for a specific token.
19             (get-token-uri (uint) (response (optional (string-ascii 256)) uint))
20
21             ;; Transfer from one principal to another.
22             (transfer (uint uint principal principal) (response bool uint))
23
24             ;; Transfer from one principal to another with a memo.
25             (transfer-memo (uint uint principal principal (buff 34)) (response bool uint))
26         )
27     )
```


 **mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week

9 lines (8 loc) · 400 Bytes

Code

Blame

Raw

```
1  (define-trait sip013-transfer-many-trait
2      (
3          ;; Transfer many tokens at once.
4          (transfer-many ((list 200 {token-id: uint, amount: uint, sender: principal, r
5
6          ;; Transfer many tokens at once with memos.
7          (transfer-many-memo ((list 200 {token-id: uint, amount: uint, sender: princip
8      )
9  )
```




main ▾

[bigmarket-dao](#) / [contracts](#) / [proposals](#) / [testnet](#)
/ bdp000-bootstrap.clar

Go to file

**mijoco-btc** Release 2. Changes include new soul bound sft based reputation system

9f68c60 · 5 days ago

Executable File · 118 lines (100 loc) · 6.99 KB

Code

Blame

Raw



```
1 ;; Title: BDP000 Bootstrap
2 ;; Description:
3 ;; Sets up and configure the DAO
4
5 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTKV9T2XNXDB908Z.proposal-trait.proposal-trait)
6
7 (define-constant token-supply u100000000000000)
8
9 (define-public (execute (sender principal))
10   (begin
11     ;; Enable genesis extensions.
12     (try! (contract-call? .bigmarket-dao set-extensions
13       (list
14         {extension: .bme000-0-governance-token, enabled: true}
15         {extension: .bme001-0-proposal-voting, enabled: true}
16         {extension: .bme003-0-core-proposals, enabled: true}
17         {extension: .bme004-0-core-execute, enabled: true}
18         {extension: .bme006-0-treasury, enabled: true}
19         {extension: .bme010-0-liquidity-contribution, enabled: true}
20         {extension: .bme021-0-market-voting, enabled: true}
21         {extension: .bme022-0-market-gating, enabled: true}
22         {extension: .bme023-0-market-predicting, enabled: true}
23         {extension: .bme023-0-market-scalar-pyth, enabled: true}
24         {extension: .bme023-0-market-bitcoin, enabled: true}
25         {extension: .bme030-0-reputation-token, enabled: true}
26       )
27     ))
28     ;; Set core team members.
29     (try! (contract-call? .bme003-0-core-proposals set-core-team-member 'ST167Z6V
30     (try! (contract-call? .bme003-0-core-proposals set-core-team-member 'ST105HCS
31
32     ;; Set executive team members.
33     (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST1SV
34     (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST2F4
35     (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST167
36     (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST105
37     (try! (contract-call? .bme004-0-core-execute set-signals-required u1)) ;; sig
```

```

38
39 ;; configure prediction markets
40 ;; allowedCreators = ["ST1SV7MYKRKKDG8PHSSKZ0W66DPKRPB5KV8ACN62G", "ST2F4ZBBV
41 (try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b
42 (try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b
43 (try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b
44
45 (try! (contract-call? .bme023-0-market-predicting set-resolution-agent 'ST167
46 (try! (contract-call? .bme023-0-market-predicting set-dev-fund 'ST1EEDB05014J
47 (try! (contract-call? .bme023-0-market-predicting set-dao-treasury .bme006-0-
48 (try! (contract-call? .bme023-0-market-predicting set-allowed-token 'ST31A25Y
49 (try! (contract-call? .bme023-0-market-predicting set-allowed-token .bme000-0
50 (try! (contract-call? .bme023-0-market-predicting set-allowed-token 'ST1F7QA2
51
52 (try! (contract-call? .bme023-0-market-scalar-pyth set-resolution-agent 'ST16
53 (try! (contract-call? .bme023-0-market-scalar-pyth set-dev-fund 'ST1EEDB05014
54 (try! (contract-call? .bme023-0-market-scalar-pyth set-dao-treasury .bme006-0
55 (try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token 'ST31A25
56 (try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token .bme000-
57 (try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token 'ST1F7QA
58
59 (try! (contract-call? .bme023-0-market-bitcoin set-resolution-agent 'ST167Z6W
60 (try! (contract-call? .bme023-0-market-bitcoin set-dev-fund 'ST1EEDB05014JVXS
61 (try! (contract-call? .bme023-0-market-bitcoin set-dao-treasury .bme006-0-tre
62
63 (try! (contract-call? .bme010-0-token-sale initialize-ido))
64
65 ;; core team voting rights unlock over u105120 bitcoin block period
66 (try! (contract-call? .bme000-0-governance-token set-core-team-vesting
67     (list
68         {recipient: sender, start-block: burn-block-height, duration:
69         {recipient: 'ST2F4ZBBV22RF2WYR424HKX5RDN6XRK19X37YEVGG, start
70         {recipient: 'STNNX8QFM2MPJ35V9SNH6BMWYRJF8KVZC3XDZGVZ, start-
71         {recipient: 'ST105HCS1RTR7D61EZET8CWNEF24ENEN3V6ARBYBJ, start
72         {recipient: 'ST167Z6WFHMOV0FZKFCRNWZ33WTB0DFBCW9M1FW3AY, start
73     )
74 ))
75 (try! (contract-call? .bme000-0-governance-token bmg-mint-many
76     (list
77         {amount: (/ (* u1500 token-supply) u10000), recipient: .bme00
78     )
79 ))
80
81 ;; Entry levels (weight: 1)
82 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u1 u1))
83 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u2 u1))
84 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u3 u1))
85
86 ;; Contributor levels (weight: 2)
87 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u4 u2))
88 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u5 u2))
89 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u6 u2))
90
91 ;; Active community (weight: 3)
92 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u7 u3))
93 (try! (contract-call? .bme030-0-reputation-token set-tier-weight u8 u3))

```



```

94      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u9 u3))
95
96      ;; Project leads (weight: 5)
97      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u10 u5))
98      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u11 u5))
99      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u12 u5))
100
101      ;; Strategic contributors (weight: 8)
102      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u13 u8))
103      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u14 u8))
104      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u15 u8))
105
106      ;; Core stewards (weight: 13)
107      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u16 u13))
108      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u17 u13))
109      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u18 u13))
110
111      ;; Founders / exec level (weight: 21)
112      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u19 u21))
113      (try! (contract-call? .bme030-0-reputation-token set-tier-weight u20 u21))
114
115      (print "BigMarket DAO has risen.")
116      (ok true)
117    )
118  )

```

[main](#) **bigmarket-dao / contracts / extensions**
/ bme000-0-governance-token.clar**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



Executable File · 268 lines (220 loc) · 8.35 KB

Code

Blame

Raw



```
1 ;; Title: BME00 Governance Token
2 ;; Synopsis:
3 ;; This extension defines the governance token of BigMarket DAO.
4 ;; Description:
5 ;; The governance token is a simple SIP010-compliant fungible token
6 ;; with some added functions to make it easier to manage by
7 ;; BigMarket DAO proposals and extensions.
8 ;; The operations vesting schedule and recipients can be updated (see current-key and
9 ;; set-core-team-vesting) up till the first claim. If more recipients are added they
10 ;; allocation is proportionally diluted.
11
12 (impl-trait .governance-token-trait.governance-token-trait)
13 (impl-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-010-trait)
14 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
15
16 (define-fungible-token bmg-token u100000000000000)
17 (define-fungible-token bmg-token-locked)
18
19 (define-constant core-team-max-vesting u15000000000000) ;; 15% of total supply (10,000,000 BIC)
20
21 (define-constant err-unauthorised (err u3000))
22 (define-constant err-not-token-owner (err u3001))
23 (define-constant err-not-core-team (err u3002))
24 (define-constant err-no-vesting-schedule (err u3003))
25 (define-constant err-nothing-to-claim (err u3004))
26 (define-constant err-core-vesting-limit (err u3005))
27 (define-constant err-cliff-not-reached (err u3006))
28 (define-constant err-recipients-are-locked (err u3007))
29 (define-constant err-transfers-blocked (err u3008))
30
31 (define-data-var token-name (string-ascii 32) "BigMarket Governance Token")
32 (define-data-var token-symbol (string-ascii 10) "BIG")
33 (define-data-var token-uri (optional (string-utf8 256)) none)
34 (define-data-var token-decimals uint u6)
35 (define-data-var core-team-size uint u0)
36
37 (define-data-var token-price uint u100000)
38 (define-data-var transfers-active bool false)
39
```

```

40 (define-map core-team-vesting-tracker principal uint) ;; Tracks vested amount per recipient
41
42 ;; ---- Vesting Storage ----
43 (define-data-var claim-made bool false)
44 (define-data-var current-key uint u0)
45 (define-map core-team-vesting {current-key: uint, recipient: principal}
46   {total-amount: uint, start-block: uint, duration: uint, claimed: uint}
47 )
48
49 ;; --- Authorisation check
50
51 (define-public (is-dao-or-extension)
52   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
53 )
54 )
55 ;; ---- Vesting Methods ----
56
57 ;; --- Vesting logic and sale
58 (define-public (set-transfers-active (new-transfers-active bool))
59   (begin
60     (try! (is-dao-or-extension))
61     (var-set transfers-active new-transfers-active)
62     (ok true)
63   )
64 )
65 (define-read-only (get-transfers-active) (var-get transfers-active))
66
67 (define-public (set-token-price (new-token-price uint))
68   (begin
69     (try! (is-dao-or-extension))
70     (var-set token-price new-token-price)
71     (ok true)
72   )
73 )
74
75 (define-public (set-core-team-vesting (core-team (list 200 {recipient: principal, start-block:
76   (begin
77     (try! (is-dao-or-extension))
78     (asserts! (not (var-get claim-made)) err-recipients-are-locked)
79     (var-set current-key (+ u1 (var-get current-key)))
80     (var-set core-team-size (len core-team))
81     (as-contract (fold set-core-team-vesting-iter core-team (ok true)))
82   )
83 )
84 (define-private (set-core-team-vesting-iter (item {recipient: principal, start-block: uint, c
85   (begin
86     (try! previous-result)
87     ;;(asserts! (as-contract (contract-call? .bme004-0-core-execute is-executive-
88     (let (
89       (amount (/ core-team-max-vesting (var-get core-team-size)))
90     )
91     (map-set core-team-vesting {current-key: (var-get current-key), recip
92       {total-amount: amount, start-block: (get start-block item), c
93     (map-set core-team-vesting-tracker (get recipient item) amount)
94     (print {event: "set-core-team-vesting", amount: amount, start-block:
95     (ok true)

```

```
96         )
97     )
98 )
99
100 (define-public (core-claim)
101   (let
102     (
103       (vesting (unwrap! (map-get? core-team-vesting {current-key: (var-get current-key), re
104       (current-block burn-block-height)
105         (start-block (get start-block vesting))
106         (duration (get duration vesting))
107         (total-amount (get total-amount vesting))
108         (claimed (get claimed vesting))
109         (elapsed (if (> current-block start-block) (- current-block start-block) u0))
110         (vested (if (> elapsed duration) total-amount (/ (* total-amount elapsed) dur
```

```
195
196 (define-public (set-token-uri (new-uri (optional (string-utf8 256))))
197   (begin
198     (try! (is-dao-or-extension))
199     (ok (var-set token-uri new-uri))
200   )
201 )
202
203 (define-private (bmg-mint-many-iter (item {amount: uint, recipient: principal}))
204   (ft-mint? bmg-token (get amount item) (get recipient item))
205 )
206
207 (define-public (bmg-mint-many (recipients (list 200 {amount: uint, recipient: principal})))
```

```

208         (begin
209             (try! (is-dao-or-extension))
210             (ok (map bmg-mint-many-iter recipients))
211         )
212     )
213
214 ;; --- Public functions
215
216 ;; sip-010-trait
217
218 (define-public (transfer (amount uint) (sender principal) (recipient principal) (memo (option
219     (begin
220         (asserts! (or (is-eq tx-sender sender) (is-eq contract-caller sender)) err-no
221         (asserts! (or (var-get transfers-active) (unwrap! (is-dao-or-extension) err-unauthori
222             (ft-transfer? bmg-token amount sender recipient)
223         )
224     )
225
226 (define-read-only (get-name)
227     (ok (var-get token-name))
228 )
229
230 (define-read-only (get-symbol)
231     (ok (var-get token-symbol))
232 )
233
234 (define-read-only (get-decimals)
235     (ok (var-get token-decimals))
236 )
237
238 (define-read-only (get-balance (who principal))
239     (ok (+ (ft-get-balance bmg-token who) (ft-get-balance bmg-token-locked who)))
240 )
241
242 (define-read-only (get-total-supply)
243     (ok (+ (ft-get-supply bmg-token) (ft-get-supply bmg-token-locked)))
244 )
245
246 (define-read-only (get-token-uri)
247     (ok (var-get token-uri))
248 )
249
250 ;; governance-token-trait
251
252 (define-read-only (bmg-get-balance (who principal))
253     (get-balance who)
254 )
255
256 (define-read-only (bmg-has-percentage-balance (who principal) (factor uint))
257     (ok (>= (* (unwrap-panic (get-balance who)) factor) (* (unwrap-panic (get-total-suppl
258 )
259
260 (define-read-only (bmg-get-locked (owner principal))
261     (ok (ft-get-balance bmg-token-locked owner))
262 )
263

```

```
264      ;; --- Extension callback
265
266      (define-public (callback (sender principal) (memo (buff 34)))
267        (ok true)
268      )
```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme001-0-proposal-voting.clar](#)[Go to file](#)**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



248 lines (219 loc) · 9.83 KB

Code

Blame

Raw



```
1 ;; Title: BME01 Proposal Voting
2 ;; Synopsis:
3 ;; Allows governance token holders to vote on and conclude proposals.
4 ;; Description:
5 ;; Once proposals are submitted, they are open for voting after a lead up time
6 ;; passes. Any token holder may vote on an open proposal, where one token equals
7 ;; one vote. Members can vote until the voting period is over. After this period
8 ;; anyone may trigger a conclusion. The proposal will then be executed if the
9 ;; votes in favour exceed the ones against by the custom majority if set or simple majority
10 ;; otherwise. Votes may additionally be submitted as batched list of signed structured
11 ;; voting messages using SIP-018.
12 ;; The mechanism for voting requires Governance tokens to be burned in exchange for the
13 ;; equivalent number of lock tokens – these can be re-exchanged after the vote is concluded.
14
15 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
16 (use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-trait)
17
18 (define-constant err-unauthorised (err u3000))
19 (define-constant err-proposal-already-executed (err u3002))
20 (define-constant err-proposal-already-exists (err u3003))
21 (define-constant err-unknown-proposal (err u3004))
22 (define-constant err-proposal-already-concluded (err u3005))
23 (define-constant err-proposal-inactive (err u3006))
24 (define-constant err-proposal-not-concluded (err u3007))
25 (define-constant err-no-votes-to-return (err u3008))
26 (define-constant err-end-block-height-not-reached (err u3009))
27 (define-constant err-disabled (err u3010))
28 (define-constant err-not-majority (err u3011))
29
30 (define-constant structured-data-prefix 0x534950303138)
31 (define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
32   {
33     name: "BigMarket",
34     version: "1.0.0",
35     chain-id: chain-id
36   }
37   ) err-unauthorised)
38 ))
39 (define-constant custom-majority-upper u10000)
```



```

39 (define-constant custom-majority upper u10000)
40 (define-constant structured-data-header (concat structured-data-prefix message-domain-hash))
41
42 (define-map proposals
43   principal
44   {
45     custom-majority: (optional uint), ;; u10000 = 100%
46     votes-for: uint,
47     votes-against: uint,
48     start-burn-height: uint,
49     end-burn-height: uint,
50     concluded: bool,
51     passed: bool,
52     proposer: principal
53   }
54 )
55
56 (define-map member-total-votes {proposal: principal, voter: principal} uint)
57
58 ;; --- Authorisation check
59
60 (define-public (is-dao-or-extension)
61   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
62   )
63
64   ;; --- Internal DAO functions
65
66   ;; Proposals
67
68   (define-public (add-proposal (proposal <proposal-trait>) (data {start-burn-height: uint, end-
69     (begin
70       (try! (is-dao-or-extension))
71       (asserts! (is-none (contract-call? .bigmarket-dao executed-at proposal)) err-
72       (asserts! (match (get custom-majority data) majority (> majority u5000) true)
73       (print {event: "propose", proposal: proposal, proposer: tx-sender}))
74       (ok (asserts! (map-insert proposals (contract-of proposal) (merge {votes-for:
75     )
76   )
77
78   ;; --- Public functions
79
80   ;; Proposals
81
82   (define-read-only (get-proposal-data (proposal principal))
83     (map-get? proposals proposal)
84   )
85
86   ;; Votes
87
88   (define-read-only (get-current-total-votes (proposal principal) (voter principal))
89     (default-to u0 (map-get? member-total-votes {proposal: proposal, voter: voter}))
90   )
91
92   (define-public (vote (amount uint) (for bool) (proposal principal) (reclaim-proposal (optional
93     (process-vote-internal amount for proposal tx-sender reclaim-proposal)
94   )
95

```

[illegible]

```

175         )
176     )
177
178     ;; Conclusion
179
180     (define-public (conclude (proposal <proposal-trait>))
181         (let
182             (
183                 (proposal-data (unwrap! (map-get? proposals (contract-of proposal)) err-unknown-proposal))
184                 (passed
185                     (match (get custom-majority proposal-data)
186                         majority (> (* (get votes-for proposal-data) custom-majority)
187                                     (> (get votes-for proposal-data) (get votes-against proposal-data))))
188                 )
189             )
190         )
191         (asserts! (not (get concluded proposal-data)) err-proposal-already-concluded)
192         (asserts! (>= burn-block-height (get end-burn-height proposal-data)) err-end-burn-height)
193         (map-set proposals (contract-of proposal) (merge proposal-data {concluded: true}))
194         (print {event: "conclude", proposal: proposal, passed: passed})
195         (and passed (try! (contract-call? .bigmarket-dao execute proposal tx-sender)))
196         (try! (contract-call? .bme030-0-reputation-token mint tx-sender u3 u5))
197         (ok passed)
198     )
199 )
200
201     ;; Reclamation
202
203     (define-public (reclaim-votes (proposal (optional principal)))
204         (let
205             (
206                 (reclaim-proposal (unwrap! proposal err-unknown-proposal))
207                 (proposal-data (unwrap! (map-get? proposals reclaim-proposal) err-unknown-proposal))

```

```

208         (votes (unwrap! (map-get? member-total-votes {proposal: reclaim-propos
209             )
210             (asserts! (get concluded proposal-data) err-proposal-not-concluded)
211             (map-delete member-total-votes {proposal: reclaim-proposal, voter: tx-sender})
212             (try! (contract-call? .bme030-0-reputation-token mint tx-sender u5 u3))
213             (contract-call? .bme000-0-governance-token bmg-unlock votes tx-sender)
214         )
215     )
216
217     ;; --- Extension callback
218
219     (define-public (callback (sender principal) (memo (buff 34)))
220         (ok true)
221     )
222
223     (define-read-only (verify-signature (hash (buff 32)) (signature (buff 65)) (signer principal)
224         (is-eq (principal-of? (unwrap! (secp256k1-recover? hash signature) false)) (ok signer)
225     )
226
227     (define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
228         (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
229     )
230
231     (define-read-only (verify-signed-tuple
232         (message-data (tuple
233             (attestation (string-ascii 100))
234             (proposal principal)
235             (vote bool)
236             (voter principal)
237             (amount uint)))
238         (signature (buff 65))
239         (signer principal))
240     (let
241         (
242             ;; Compute the structured data hash
243             (structured-data-hash (sha256 (unwrap! (to-consensus-buff? message-data) err-unauthor
244         )
245         ;; Verify the signature using the computed hash
246         (ok (verify-signed-structured-data structured-data-hash signature signer))
247     )
248 )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme003-0-core-proposals.clar](#)[Go to file](#) **mijoco-btc** attempt to resolve clarinet toml issues around pyth. code is basicall... [...](#)767bb51 · 3 weeks ago 

Executable File · 73 lines (59 loc) · 2.53 KB

[Code](#) [Blame](#)[Raw](#)

```
1 ;; Title: BME03 Core Proposals
2 ;; Synopsis:
3 ;; This extension allows for the creation of core proposals by a few trusted
4 ;; principals.
5 ;; Description:
6 ;; Only a list of trusted principals, designated as the
7 ;; "core team", can create core proposals. The core proposal
8 ;; extension has an optional ~3 month sunset period, after which no more core
9 ;; proposals can be made – set it to 0 to disable. The core team members, sunset period, and
10 ;; core vote duration can be changed by means of a future proposal.
11
12 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
13 (use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-t
14
15 (define-data-var core-team-sunset-height uint u0) ;; does not expire by default – can be char
16
17 (define-constant err-unauthorised (err u3300))
18 (define-constant err-not-core-team-member (err u3301))
19 (define-constant err-sunset-height-reached (err u3302))
20 (define-constant err-sunset-height-in-past (err u3303))
21
22 (define-map core-team principal bool)
23
24 ;; --- Authorisation check
25
26 (define-public (is-dao-or-extension)
27   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
28   )
29
30 ;; --- Internal DAO functions
31
32 (define-public (set-core-team-sunset-height (height uint))
33   (begin
34     (try! (is-dao-or-extension))
35     (asserts! (> height burn-block-height) err-sunset-height-in-past)
36     (ok (var-set core-team-sunset-height height))
37   )
```

```

38 )
39
40 (define-public (set-core-team-member (who principal) (member bool))
41   (begin
42     (try! (is-dao-or-extension))
43     (print {event: "set-core-team-member", who: who, member: member})
44     (ok (map-set core-team who member))
45   )
46 )
47
48 ;; --- Public functions
49
50 (define-read-only (is-core-team-member (who principal))
51   (default-to false (map-get? core-team who))
52 )
53
54 (define-public (core-propose (proposal <proposal-trait>) (start-burn-height uint) (duration u
55   (begin
56     (asserts! (is-core-team-member tx-sender) err-not-core-team-member)
57     (asserts! (or (is-eq (var-get core-team-sunset-height) u0) (< burn-block-heig
58     (contract-call? .bme001-0-proposal-voting add-proposal proposal
59       {
60         start-burn-height: start-burn-height,
61         end-burn-height: (+ start-burn-height duration),
62         custom-majority: custom-majority,
63         proposer: tx-sender ;; change to original submitter
64       }
65     )
66   )
67 )
68
69 ;; --- Extension callback
70
71 (define-public (callback (sender principal) (memo (buff 34)))
72   (ok true)
73 )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme004-0-core-execute.clar](#)

...

 **mijoco-btc** attempt to resolve clarinet toml issues around pyth. code is basicall... 767bb51 · 3 weeks ago 

Executable File · 99 lines (80 loc) · 3.62 KB

[Code](#) [Blame](#)

Raw



```
1 ;; Title: BME04 Core Execute
2 ;; Synopsis:
3 ;; This extension allows a small number of very trusted principals to immediately
4 ;; execute a proposal once a super majority is reached.
5 ;; Description:
6 ;; An extension meant for the bootstrapping period of a DAO. It temporarily gives
7 ;; some very trusted principals the ability to perform an "executive action";
8 ;; meaning, they can skip the voting process to immediately executive a proposal.
9 ;; The Core execute extension has an optional sunset period of ~1 month from deploy
10 ;; time, set it to 0 to disable. The core executive team, parameters, and sunset period may b
11 ;; by means of a future proposal.
12
13 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTK9T2XNXDB908Z.extension-trait.extension-trait)
14 (use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTK9T2XNXDB908Z.proposal-trait.proposal-
15
16 (define-data-var executive-team-sunset-height uint u0) ;; does not expire by default - can be
17
18 (define-constant err-unauthorised (err u3400))
19 (define-constant err-not-executive-team-member (err u3401))
20 (define-constant err-already-executed (err u3402))
21 (define-constant err-sunset-height-reached (err u3403))
22 (define-constant err-sunset-height-in-past (err u3404))
23
24 (define-map executive-team principal bool)
25 (define-map executive-action-signals {proposal: principal, team-member: principal} bool)
26 (define-map executive-action-signal-count principal uint)
27
28 (define-data-var executive-signals-required uint u1) ;; signals required for an executive act
29
30 ;; --- Authorisation check
31
32 (define-public (is-dao-or-extension)
33   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
34 )
35
36 ;; --- Internal DAO functions
37
```

```

38     (define-public (set-executive-team-sunset-height (height uint))
39         (begin
40             (try! (is-dao-or-extension))
41             (asserts! (> height burn-block-height) err-sunset-height-in-past)
42             (ok (var-set executive-team-sunset-height height))
43         )
44     )
45
46     (define-public (set-executive-team-member (who principal) (member bool))
47         (begin
48             (try! (is-dao-or-extension))
49             (ok (map-set executive-team who member))
50         )
51     )
52
53     (define-public (set-signals-required (new-requirement uint))
54         (begin
55             (try! (is-dao-or-extension))
56             (ok (var-set executive-signals-required new-requirement))
57         )
58     )
59
60     ;; --- Public functions
61
62     (define-read-only (is-executive-team-member (who principal))
63         (default-to false (map-get? executive-team who))
64     )
65
66     (define-read-only (has-signalled (proposal principal) (who principal))
67         (default-to false (map-get? executive-action-signals {proposal: proposal, team-member:
68     }
69
70     (define-read-only (get-signals-required)
71         (var-get executive-signals-required)
72     )
73
74     (define-read-only (get-signals (proposal principal))
75         (default-to u0 (map-get? executive-action-signal-count proposal))
76     )
77
78     (define-public (executive-action (proposal <proposal-trait>))
79         (let
80             (
81                 (proposal-principal (contract-of proposal))
82                 (signals (+ (get-signals proposal-principal) (if (has-signalled propo
83             )
84             (asserts! (is-executive-team-member tx-sender) err-not-executive-team-member)
85             (asserts! (or (is-eq (var-get executive-team-sunset-height) u0) (< burn-block
86             (and (>= signals (var-get executive-signals-required))
87                 (try! (contract-call? .bigmarket-dao execute proposal tx-sender)))
88             )
89             (map-set executive-action-signals {proposal: proposal-principal, team-member:
90             (map-set executive-action-signal-count proposal-principal signals)
91             (ok signals)
92         )
93     )

```



```
94
95 ;; --- Extension callback
96
97 (define-public (callback (sender principal) (memo (buff 34)))
98   (ok true)
99 )
```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme006-0-treasury.clar](#)

...

 **mijoco-btc** attempt to resolve clarinet toml issues around pyth. code is basicall... ...767bb51 · 3 weeks ago 

Executable File · 155 lines (126 loc) · 4.92 KB

[Code](#) [Blame](#)[Raw](#)

```
1  ;; Title: BME006 Treasury
2  ;; Synopsis:
3  ;; A treasury that can manage STX, SIP009, SIP010, and SIP013 tokens.
4  ;; Description:
5  ;; An extension contract that is meant to hold tokens on behalf of the
6  ;; DAO. It can hold and transfer STX, SIP009, SIP010, and SIP013 tokens.
7  ;; They can be deposited by simply transferring them to the contract.
8  ;; Any extension or executing proposal can trigger transfers.
9  ;; Technically, the ExecutorDAO core can hold and transfer tokens
10 ;; directly. The treasury extension merely adds a bit of separation.
11
12 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
13
14 (define-constant err-unauthorised (err u3000))
15
16 ;; --- Transferable traits
17
18 (define-trait sip009-transferable
19   (
20     (transfer (uint principal principal) (response bool uint))
21   )
22 )
23
24 (define-trait sip010-transferable
25   (
26     (transfer (uint principal principal (optional (buff 34))) (response bool uint)
27   )
28 )
29
30 (define-trait sip013-transferable
31   (
32     (transfer (uint uint principal principal) (response bool uint))
33     (transfer-memo (uint uint principal principal (buff 34)) (response bool uint)
34   )
35 )
36
37 (define-trait sip013-transferable-many
```

```

38      (
39          (transfer-many ((list 200 {token-id: uint, amount: uint, sender: principal, r
40              (transfer-many-memo ((list 200 {token-id: uint, amount: uint, sender: princip
41      )
42  )
43
44  ;; --- Authorisation check
45
46  (define-public (is-dao-or-extension)
47      (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
48  )
49
50  ;; --- Internal DAO functions
51
52  ;; STX
53
54  (define-public (stx-transfer (amount uint) (recipient principal) (memo (optional (buff 34))))
55      (begin
56          (try! (is-dao-or-extension))
57          (match memo to-print (print to-print) 0x)
58          (as-contract (stx-transfer? amount tx-sender recipient))
59      )
60  )
61
62  (define-public (stx-transfer-many (transfers (list 200 {amount: uint, recipient: principal, n
63      (begin
64          (try! (is-dao-or-extension))
65          (as-contract (fold stx-transfer-many-iter transfers (ok true)))
66      )
67  )
68
69  ;; SIP009
70
71  (define-public (sip009-transfer (token-id uint) (recipient principal) (asset <sip009-transfer
72      (begin
73          (try! (is-dao-or-extension))
74          (as-contract (contract-call? asset transfer token-id tx-sender recipient))
75      )
76  )
77
78  (define-public (sip009-transfer-many (data (list 200 {token-id: uint, recipient: principal}))
79      (begin
80          (as-contract (fold sip009-transfer-many-iter data asset))
81          (ok true)
82      )
83  )
84
85  ;; SIP010
86
87  (define-public (sip010-transfer (amount uint) (recipient principal) (memo (optional (buff 34)
88      (begin
89          (try! (is-dao-or-extension))
90          (as-contract (contract-call? asset transfer amount tx-sender recipient memo))
91      )
92  )
93

```

```


94     (define-public (sip010-transfer-many (data (list 200 {amount: uint, recipient: principal, memo: (opt
95         (begin
96             (as-contract (fold sip010-transfer-many-iter data asset))
97             (ok true)
98         )
99     )
100
101 ;; SIP013
102
103 (define-public (sip013-transfer (token-id uint) (amount uint) (recipient principal) (memo (opt
104     (begin
105         (try! (is-dao-or-extension))
106         (as-contract (match memo memo-buff
107             (contract-call? asset transfer-memo token-id amount tx-sender recipient)
108             (contract-call? asset transfer token-id amount tx-sender recipient)
109         ))
110     )
111 )
112
113 (define-public (sip013-transfer-many (transfers (list 200 {token-id: uint, amount: uint, sender: principal, memo: (opt
114     (begin
115         (try! (is-dao-or-extension))
116         (as-contract (contract-call? asset transfer-many transfers))
117     )
118 )
119
120 (define-public (sip013-transfer-many-memo (transfers (list 200 {token-id: uint, amount: uint, sender: principal, memo: (opt
121     (begin
122         (try! (is-dao-or-extension))
123         (as-contract (contract-call? asset transfer-many-memo transfers))
124     )
125 )
126
127 ;; --- Iterator functions
128
129 (define-private (stx-transfer-many-iter (data {amount: uint, recipient: principal, memo: (opt
130     (begin
131         (try! previous-result)
132         (match (get memo data) to-print (print to-print) 0x)
133         (stx-transfer? (get amount data) tx-sender (get recipient data))
134     )
135 )
136
137 (define-private (sip009-transfer-many-iter (data {token-id: uint, recipient: principal}) (asset
138     (begin
139         (unwrap-panic (contract-call? asset transfer (get token-id data) tx-sender (get memo data)
140             asset)
141     )
142 )
143
144 (define-private (sip010-transfer-many-iter (data {amount: uint, recipient: principal, memo: (opt
145     (begin
146         (unwrap-panic (contract-call? asset transfer (get amount data) tx-sender (get memo data)
147             asset)
148     )
149 )

```

```
150
151 ;; --- Extension callback
152
153 (define-public (callback (sender principal) (memo (buff 34)))
154   (ok true)
155 )
```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme010-0-liquidity-contribution.clar](#)

...

 **mijoco-btc** Release 2. Changes include new soul bound sft based reputation system9f68c60 · 5 days ago 

Executable File · 62 lines (51 loc) · 1.96 KB

[Code](#) [Blame](#)[Raw](#)

```
1 ;; Title: BME010 Reputation-Gated Liquidity Contribution
2 ;; Synopsis:
3 ;; Accept STX from contributors and reward them with BGR reputation tokens
4 ;; Description:
5 ;; Users are rewarded with BGR by contributing STX to the DAO treasury.
6 ;; BGR is used to claim BIG through the main reputation contract.
7 ;; The rate is set by the DAO and can be updated as needed.
8
9 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTK9T2XNXDB908Z.extension-trait.extension-trait)
10
11 ;; Constants and Errors
12 (define-constant err-unauthorised (err u5000))
13 (define-constant err-zero-amount (err u5001))
14
15 ;; Contract variables
16 (define-data-var stx-to-bigr-rate uint u10) ;; Default: 1 STX = 10 BGR
17
18 (define-map stx-contributions {who: principal} uint)
19
20 ;; Authorization check
21 (define-public (is-dao-or-extension)
22   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-extens
23   )
24
25 ;; DAO can update the reward rate
26 (define-public (set-liquidity-reward-rate (new-rate uint))
27   (begin
28     (try! (is-dao-or-extension))
29     (var-set stx-to-bigr-rate new-rate)
30     (ok true)
31   )
32 )
33 (define-read-only (get-liquidity-reward-rate)
34   (var-get stx-to-bigr-rate)
35 )
36
37 (define-public (contribute-stx (amount uint))
```

```

38     (let (
39         (user tx-sender)
40         (rate (var-get stx-to-bigr-rate))
41         (bigr-earned (* amount rate))
42         (existing (default-to u0 (map-get? stx-contributions {who: user})))
43     )
44     (asserts! (> amount u0) err-zero-amount)
45
46     ;; Transfer STX to the DAO treasury
47     (try! (stx-transfer? amount user .bme006-0-treasury))
48
49     ;; Record contribution
50     (map-set stx-contributions {who: user} (+ existing amount))
51
52     ;; Mint BIGR to the contributor
53     (try! (contract-call? .bme030-0-reputation-token mint user u7 bigr-earned))
54     (print {event: "liquidity_contribution", from: user, amount: amount, bigr: bigr-earned})
55     (ok bigr-earned)
56 )
57 )
58
59 ;; Extension trait callback stub
60 (define-public (callback (sender principal) (memo (buff 34))))
61 (ok true)
62 )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme010-0-token-sale.clar](#)

...

**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



Executable File · 175 lines (142 loc) · 6.56 KB

Code

Blame

Raw



```
1 ;; Title: BME010 Token Sale
2 ;; Synopsis:
3 ;; Enables token sale for governance tokens.
4 ;; Description:
5 ;; Allows to token sale over 6 stages with token price set at each stage by the current DAO.
6 ;; Contract allows any stage to be cancelled and for tokens to be reclaimed.
7 ;; Listing via a DEX is not supported but can be enabled at any stage
8 ;;
9
10 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTK9T2XNXDB908Z.extension-trait.extension-trait)
11
12 (define-constant err-unauthorised (err u5000))
13 (define-constant err-invalid-stage (err u5001))
14 (define-constant err-stage-sold-out (err u5002))
15 (define-constant err-nothing-to-claim (err u5003))
16 (define-constant err-no-more-stages (err u5005))
17 (define-constant err-already-cancelled (err u5006))
18 (define-constant err-no-purchase (err u5007))
19 (define-constant err-stage-not-cancelled (err u5008))
20 (define-constant err-stage-cancelled (err u5009))
21
22 (define-data-var current-stage uint u1) ;; IDO starts at Stage 1
23 (define-data-var current-stage-start uint burn-block-height) ;; Tracks burn-block-height when
24
25 (define-map ido-stage-details uint
26   {price: uint, max-supply: uint, tokens-sold: uint, cancelled: bool})
27 (define-map ido-purchases {stage: uint, buyer: principal} uint) ;; Tracks purchases
28
29 ;; --- Authorisation check
30
31 (define-public (is-dao-or-extension)
32   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
33   )
34
35 (define-read-only (get-ido-stages)
36   (list
37     (map-get? ido-stage-details u1)
38     (map-get? ido-stage-details u2)
39     (map-get? ido-stage-details u3))
```



```

39      (map-get? ido-stage-details u5)
40      (map-get? ido-stage-details u4)
41      (map-get? ido-stage-details u5)
42      (map-get? ido-stage-details u6)
43    )
44  )
45
46  (define-read-only (get-ido-user-for-stage (stage uint) (who principal))
47    (map-get? ido-purchases {stage: stage, buyer: who}))
48  )
49
50  (define-read-only (get-ido-user (who principal))
51    (list
52      (match (map-get? ido-purchases {stage: u1, buyer: who}) value value u0 )
53      (match (map-get? ido-purchases {stage: u2, buyer: who}) value value u0 )
54      (match (map-get? ido-purchases {stage: u3, buyer: who}) value value u0 )
55      (match (map-get? ido-purchases {stage: u4, buyer: who}) value value u0 )
56      (match (map-get? ido-purchases {stage: u5, buyer: who}) value value u0 )
57    )
58  )
59
60  ;; --- Internal DAO functions
61
62  (define-public (initialize-ido)
63    (begin
64      (try! (is-dao-or-extension))
65
66      ;; Set up each stage
67      (map-set ido-stage-details u1 {price: u5, max-supply: u600000000000, tokens-sold: u0, car
68      (map-set ido-stage-details u2 {price: u6, max-supply: u8333333000000, tokens-sold: u0, car
69      (map-set ido-stage-details u3 {price: u7, max-supply: u1071429000000, tokens-sold: u0, ca
70      (map-set ido-stage-details u4 {price: u8, max-supply: u1250000000000, tokens-sold: u0, ca
71      (map-set ido-stage-details u5 {price: u10, max-supply: u1500000000000, tokens-sold: u0, c
72      (map-set ido-stage-details u6 {price: u20, max-supply: u1000000000000, tokens-sold: u0, c
73
74      (print {event: "ido-initialized"})
75      (ok true)
76    )
77  )
78
79  (define-public (buy-ido-tokens (stx-amount uint))
80    (let (
81      (stage (var-get current-stage))
82      (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
83      (bmg-price (get price stage-info))
84      (max-supply (get max-supply stage-info))
85      (tokens-sold (get tokens-sold stage-info))
86      (sender tx-sender)
87      (cancelled (get cancelled stage-info))
88      (current-stake (default-to u0 (map-get? ido-purchases {stage: stage, buyer: tx-sender})))
89      (tokens-to-buy (* stx-amount bmg-price))
90    )
91
92    ;; Ensure enough supply remains
93    (asserts! (<= (+ tokens-sold tokens-to-buy) max-supply) err-stage-sold-out)
94    (asserts! (not cancelled) err-stage-cancelled)
95

```

```

96      ;; Accept STX payment
97      (try! (stx-transfer? stx-amount tx-sender .bme006-0-treasury))
98
99      ;; Mint tokens directly to the buyer
100     (try! (as-contract (contract-call? .bme000-0-governance-token bmg-mint tokens-to-buy send
101
102     ;; Update stage details
103     (map-set ido-stage-details stage (merge stage-info {tokens-sold: (+ tokens-sold tokens-to
104     (map-set ido-purchases {stage: stage, buyer: tx-sender} (+ current-stake tokens-to-buy))
105
106     (print {event: "ido-purchase", buyer: tx-sender, stage: stage, tokens: tokens-to-buy, stx
107
108     (ok tokens-to-buy)
109   )
110 )
111
112 (define-public (advance-ido-stage)
113   (begin
114     (try! (is-dao-or-extension))
115     (let (
116       (stage (var-get current-stage))
117       (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
118     )
119
120     (asserts! (not (get cancelled stage-info)) err-already-cancelled) ;; Ensure not already c
121     (asserts! (< stage u6) err-no-more-stages) ;; Can't go past stage 6
122     (var-set current-stage (+ u1 stage)) ;; Move to the next stage
123
124     ;; Use burn-block-height to track when the stage starts
125     (var-set current-stage-start burn-block-height)
126
127     (print {event: "ido-stage-advanced", new-stage: (var-get current-stage), burn-start: burn
128     (ok stage)
129   )
130
131 )
132 )
133
134 (define-public (cancel-ido-stage)
135   (begin
136     (try! (is-dao-or-extension))
137
138     (let ((stage (var-get current-stage))
139           (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage)))
140
141       (asserts! (not (get cancelled stage-info)) err-already-cancelled) ;; Ensure not already
142
143       ;; Update the stage's cancelled flag
144       (map-set ido-stage-details stage (merge stage-info {cancelled: true}))
145
146       (print {event: "cancel-ido-stage", stage: stage})
147       (ok true)
148     )
149   )
150 )
151

```

```

152 (define-public (claim-ido-refund)
153   (let ((stage (var-get current-stage))
154         (purchase-amount (unwrap! (map-get? ido-purchases {stage: stage, buyer: tx-sender}) e
155         (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
156         (price (get price stage-info))
157         (sender tx-sender)
158         )
159     ;; Ensure stage is actually cancelled
160     (asserts! (get cancelled stage-info) err-stage-not-cancelled)
161     ;; Transfer STX back to the buyer / burn the bdg
162     (try! (as-contract (contract-call? .bme006-0-treasury stx-transfer (* purchase-amount pri
163     (try! (as-contract (contract-call? .bme000-0-governance-token bmg-burn purchase-amount se
164     ;; Remove the purchase record
165     (map-delete ido-purchases {stage: stage, buyer: tx-sender}))
166     (print {event: "ido-refund", buyer: tx-sender, refunded: purchase-amount, stage: stage})
167     (ok purchase-amount)
168   )
169 )
170
171 ;; --- Extension callback
172
173 (define-public (callback (sender principal) (memo (buff 34))))
174   (ok true)
175 )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme021-0-market-voting.clar](#)

...

**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



Executable File · 324 lines (282 loc) · 12.9 KB

Code

Blame

Raw



```
1 ;; Title: BME021 Market Voting
2 ;; Synopsis:
3 ;; Intended for prediction market resolution via community voting.
4 ;; Description:
5 ;; Market votes are connected to a specific market via the market data hash and
6 ;; votes are created via challenges to the market outcome. Any user with a stake in the market
7 ;; can challenge the outcome. Voting begins on challenge and runs for a DAO configured window
8 ;; DAO governance voting resolves the market – either confirming or changing the original
9 ;; outcome based on a simple majority.
10 ;; Unlike proposal voting – market voting is categorical – voters are voting to select an
11 ;; outcome from at least 2 and up to 10 potential outcomes.
12
13 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTK9T2XNXDB908Z.extension-trait.extension-trait)
14 (use-trait nft-trait 'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
15 (use-trait ft-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
16 (use-trait prediction-market-trait .prediction-market-trait.prediction-market-trait)
17
18 (define-constant err-unauthorised (err u2100))
19 (define-constant err-poll-already-exists (err u2102))
20 (define-constant err-unknown-proposal (err u2103))
21 (define-constant err-proposal-inactive (err u2105))
22 (define-constant err-already-voted (err u2106))
23 (define-constant err-proposal-start-not-reached (err u2109))
24 (define-constant err-expecting-root (err u2110))
25 (define-constant err-invalid-signature (err u2111))
26 (define-constant err-proposal-already-concluded (err u2112))
27 (define-constant err-end-burn-height-not-reached (err u2113))
28 (define-constant err-no-votes-to-return (err u2114))
29 (define-constant err-not-concluded (err u2115))
30 (define-constant err-invalid-category (err u2116))
31
32 (define-constant structured-data-prefix 0x534950303138)
33 (define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
34   {
35     name: "BigMarket",
36     version: "1.0.0",
37     chain-id: chain-id
38   }
39   ) err-unauthorised))
```

```

39      , sha256-hash)
40    ))
41
42    (define-constant structured-data-header (concat structured-data-prefix message-domain-hash))
43
44    (define-data-var voting-duration uint u288)
45
46    (define-map resolution-polls
47      {market: principal, market-id: uint}
48      {
49        votes: (list 10 uint), ;; votes for each category. NB with 2 categories the votes at 0 are
50          end-burn-height: uint,
51          proposer: principal,
52          concluded: bool,
53          num-categories: uint,
54          winning-category: (optional uint),
55        }
56    )
57    (define-map member-total-votes {market-id: uint, voter: principal} uint)
58
59    ;; --- Authorisation check
60
61    (define-public (is-dao-or-extension)
62      (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
63    )
64
65    ;; --- Internal DAO functions
66    (define-public (set-voting-duration (new-duration uint))
67      (begin
68        (try! (is-dao-or-extension))
69        (var-set voting-duration new-duration)
70        (ok true)
71      )
72    )
73
74    ;; called by a staker in a market to begin dispute resolution process
75    (define-public (create-market-vote
76      (market <prediction-market-trait>)
77      (market-id uint)
78      (empty-votes (list 10 uint))
79      (num-categories uint)
80    )
81      (let
82        (
83          (original-sender tx-sender)
84        )
85        (asserts! (is-none (map-get? resolution-polls {market-id: market-id, market: (contract-of
86        (asserts! (is-eq (len empty-votes) num-categories) err-poll-already-exists)
87          ;; a user with stake can propose but only for a market in the correct state i
88        (try! (as-contract (contract-call? market dispute-resolution market-id original-sender)))
89
90        ;; Register the poll
91        (map-set resolution-polls {market-id: market-id, market: (contract-of market)}
92          {
93            votes: empty-votes,
94            end-burn-height: (+ burn-block-height (var-get voting-duration)),
95            proposer: tx-sender,

```

```
96         concluded: false,
97         num-categories: num-categories,
98         winning-category: none}
99     )
100
101     ;; Emit an event for the new poll
102     (print {event: "create-market-vote", market-id: market-id, proposer: tx-sender, market: n
103     (ok true)
104     )
105 )
106
107 ;; --- Public functions
108
109 (define-read-only (get-poll-data (market principal) (market-id uint))
110     (map-get? resolution-polls {market-id: market-id, market: market}))
```



```
251     )
252
253     (define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
254         (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
255     )
256
257     ;; Conclusion
258
259     (define-read-only (get-poll-status (market principal) (market-id uint))
260         (let
261             (
262                 (poll-data (unwrap! (map-get? resolution-polls {market-id: market-id, market: mar
263                 (is-active (< burn-block-height (get end-burn-height poll-data)))
```



```

264 )
265     (ok {active: is-active, concluded: (get concluded poll-data), votes: (get votes poll-
266 )
267 )
268
269
270 (define-public (conclude-market-vote (market <prediction-market-trait>) (market-id uint))
271     (let
272         (
273         (poll-data (unwrap! (map-get? resolution-polls {market-id: market-id, market: (contract
274 (votes (get votes poll-data))
275 (winning-category (get max-index (find-max-category votes)))
276 (total-votes (fold + votes u0))
277 (winning-votes (unwrap! (element-at? votes winning-category) err-already-voted))
278 (result (try! (contract-call? market resolve-market-vote market-id winning-category)))
279 )
280 (asserts! (not (get concluded poll-data)) err-proposal-already-concluded)
281 (asserts! (>= burn-block-height (get end-burn-height poll-data)) err-end-burn
282 (map-set resolution-polls {market-id: market-id, market: (contract-of market)
283 (print {event: "conclude-market-vote", market-id: market-id, winning-category
284 (try! (contract-call? .bme030-0-reputation-token mint tx-sender u3 u3))
285 (ok winning-category)
286 )
287 )
288
289 (define-public (reclaim-votes (market principal) (id (optional uint)))
290     (let
291         (
292         (market-id (unwrap! id err-unknown-proposal))
293         (poll-data (unwrap! (map-get? resolution-polls {market: market, market-id: market-id})
294 (votes (unwrap! (map-get? member-total-votes {market-id: market-id, v
295 )
296 (asserts! (get concluded poll-data) err-not-concluded)
297 (map-delete member-total-votes {market-id: market-id, voter: tx-sender}))
298 (contract-call? .bme000-0-governance-token bmg-unlock votes tx-sender)
299 )
300 )
301
302 ;; --- Extension callback
303 (define-public (callback (sender principal) (memo (buff 34)))
304     (ok true)
305 )
306
307
308 (define-private (find-max-category (votes (list 10 uint)))
309     (fold find-max-iter votes {max-votes: u0, max-index: u0, current-index: u0})
310 )
311
312 (define-private (find-max-iter (current-votes uint) (acc (tuple (max-votes uint) (max-index u
313 (let
314     (
315     (max-votes (get max-votes acc)) ;; Extract highest vote count so far
316     (max-index (get max-index acc)) ;; Extract category index with highest votes
317     (current-index (get current-index acc)) ;; Track current category index
318 )
319 (if (> current-votes max-votes)
320     (tuple (max-votes current-votes) (max-index current-index)) (current-index (current-index

```

```
320      (tuple (max-votes current-votes) (max-index current-index) (current-index (+ current-index 1)))
321      (tuple (max-votes max-votes) (max-index max-index) (current-index (+ current-index u1)))
322    )
323  )
324 )
```

[main](#) **bigmarket-dao / contracts / extensions**
/ bme022-0-market-gating.clar

...

 **mijoco-btc** attempt to resolve clarinet toml issues around pyth. code is basicall... 767bb51 · 3 weeks ago 

Executable File · 201 lines (176 loc) · 7.96 KB

[Code](#) [Blame](#)[Raw](#)

```
1 ;; Title: BME021 Market Gating
2 ;; Synopsis:
3 ;; Efficient access control using merkle proofs.
4 ;; Description:
5 ;; Provides gating functionality based on account (can-access-by-account) and
6 ;; ownership (can-access-by-ownership). The map of keys / roots are DAO managed.
7 ;; Keys can be any data hash or a specific contract id hash. For ownership the user
8 ;; must pass either an NFT or FT token and a merkle proof of ownership. For access
9 ;; by account the account principal is passed along with the proof.
10
11 ;; Define the SIP-009 and SIP-010 traits
12 (use-trait nft-trait 'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
13 (use-trait ft-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
14 (impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
15
16 (define-constant err-unauthorised (err u2200))
17 (define-constant err-either-sip9-or-sip10-required (err u2201))
18 (define-constant err-token-contract-invalid (err u2202))
19 (define-constant err-token-ownership-invalid (err u2203))
20 (define-constant err-expecting-nft-contract (err u2204))
21 (define-constant err-expecting-ft-contract (err u2205))
22 (define-constant err-expecting-token-id (err u2206))
23 (define-constant err-not-nft-owner (err u2207))
24 (define-constant err-not-ft-owner (err u2208))
25 (define-constant err-expecting-nft-buffer (err u2209))
26 (define-constant err-expecting-ft-buffer (err u2210))
27 (define-constant err-expecting-valid-merkle-proof (err u2211))
28 (define-constant err-expecting-merkle-root-for-poll (err u2212))
29 (define-constant err-expecting-an-owner (err u2213))
30 (define-constant err-account-proof-invalid (err u2214))
31 (define-constant err-ownership-proof-invalid (err u2215))
32
33 ;; Merkle roots for gated data
34 (define-map merkle-roots
35   (buff 32)
36   {
37     merkle-root: (buff 32)
```

```

38     }
39 )
40
41 (define-public (is-dao-or-extension)
42   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
43 )
44 )
45 (define-public (set-merkle-root (hashed-id (buff 32)) (root (buff 32)))
46   (begin
47     ;; Ensure only dao can set the root
48     (try! (is-dao-or-extension))
49     (map-set merkle-roots hashed-id { merkle-root: root})
50     (print {event: "merkle-root", hashed-id: hashed-id, merkle-root: (map-get? merkle-roots h
51     (ok true)
52   )
53 )
54
55 (define-public (set-merkle-root-by-principal (contract-id principal) (root (buff 32)))
56   (let
57     (
58       ;; construct the key from the contract-id
59       (principal-contract (unwrap! (principal-destruct? contract-id) (err u1001)))
60       (contract-bytes (get hash-bytes principal-contract))
61       ;; panics if not contract principal
62       (contract-name (unwrap! (to-consensus-buff? (unwrap! (get name principal-contract) er
63       (contract-key (sha256 (concat contract-bytes contract-name )))
64     )
65     ;; Ensure only dao can set the root
66     (try! (is-dao-or-extension))
67     (map-set merkle-roots contract-key { merkle-root: root})
68     (print {event: "set-merkle-root-by-principal", contract-id: contract-id, contract-name:
69     (ok true)
70   ))
71
72 (define-read-only (get-merkle-root (hashed-id (buff 32)))
73   (map-get? merkle-roots hashed-id)
74 )
75
76 ;; Verify a Merkle proof
77 (define-private (calculate-hash (hash1 (buff 32)) (hash2 (buff 32)) (position bool))
78   (if position
79     (sha256 (concat hash2 hash1))
80     (sha256 (concat hash1 hash2))
81   )
82 )
83
84 (define-private (process-proof-step (proof-step (tuple (position bool) (hash (buff 32)))) (c
85   (let ((position (get position proof-step))
86         (hash (get hash proof-step)))
87     (calculate-hash current hash position)
88   )
89 )
90
91 (define-private (verify-merkle-proof
92   (leaf (buff 32)) ;; The leaf hash (token hash)
93   (proof (list 10 (tuple (position bool) (hash (buff 32)))))

```

```

94      (root (buff 32))
95    )
96    (let
97      (
98        (calculated-root
99          (fold process-proof-step proof leaf)
100        )
101      )
102      (ok (is-eq calculated-root root))
103    )
104  )
105
106
107  (define-private (verify-nft-ownership
108    (nft-contract <nft-trait>) ;; NFT contract
109    (voter principal)          ;; Voter's principal
110    (token-id uint)            ;; Token ID
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127    )
128    (ok (>= balance quantity))
129  ))
130
131
132  ;; Validate proof of access
133  (define-public (can-access-by-ownership
134    (market-data-hash (buff 32)) ;; The hashed ID
135    (nft-contract (optional <nft-trait>)) ;; Optional NFT contract
136    (ft-contract (optional <ft-trait>)) ;; Optional FT contract
137    (token-id (optional uint)) ;; Token ID for NFTs
138    (proof (list 10 (tuple (position bool) (hash (buff 32))))) ;; The Merkle proof
139    (quantity uint) ;; Required token quantity
140  )
141  (let
142    (
143      ;; Determine if this is an NFT or FT contract
144      (is-nft-contract (is-some nft-contract))
145
146      ;; Fetch the Merkle root for the poll
147      (root (unwrap! (map-get? merkle-roots market-data-hash) err-expecting-merkle-root-for
148
149      ;; Compute the Merkle proof leaf

```

```

150      (contract-id (if is-nft-contract
151                      (unwrap! (to-consensus-buff? (as-contract (unwrap! nft-contract err-
152                      (unwrap! (to-consensus-buff? (as-contract (unwrap! ft-contract err-e
153      (leaf (sha256 contract-id))
154
155      ;; Verify the Merkle proof
156      (proof-valid (unwrap! (verify-merkle-proof leaf proof (get merkle-root root)) err-exp
157
158      ;; Verify ownership or balance
159      (ownership-valid
160        (if is-nft-contract
161          (unwrap! (verify-nft-ownership (unwrap! nft-contract err-expecting-nft-contract
162          (unwrap! (verify-ft-balance (unwrap! ft-contract err-expecting-ft-contract) tx-
163        )
164      ;; Ensure both conditions are satisfied
165      (asserts! proof-valid err-ownership-proof-invalid)
166      (asserts! ownership-valid err-token-ownership-invalid)
167      (ok true)
168    ))
169
170  (define-public (can-access-by-account
171    (sender principal)
172    (proof (list 10 (tuple (position bool) (hash (buff 32))))) ;; The Merkle proof
173  )
174  (let
175    (
176      ;; Fetch the Merkle root for the poll
177      (principal-contract (unwrap! (principal-destruct? tx-sender) (err u1001)))
178      (contract-bytes (get hash-bytes principal-contract))
179      (contract-name (unwrap! (to-consensus-buff? (unwrap! (get name principal-contract) er
180      (contract-key (sha256 (concat contract-bytes contract-name )))
181      (root (unwrap! (map-get? merkle-roots contract-key) err-expecting-merkle-root-for-pol
182
183      ;; Compute the Merkle proof leaf
184      (principal-data (unwrap! (principal-destruct? sender) (err u1001)))
185      (leaf (sha256 (get hash-bytes principal-data)))
186
187      ;; Verify the Merkle proof
188      (proof-valid (unwrap! (verify-merkle-proof leaf proof (get merkle-root root)) err-exp
189    )
190    ;; Ensure both conditions are satisfied
191    (asserts! proof-valid err-account-proof-invalid)
192    (print {event: "can-access-by-account", contract-key: contract-key, contract-name: contra
193    (ok true)
194  ))
195
196
197  ;; --- Extension callback
198  (define-public (callback (sender principal) (memo (buff 34)))
199    (ok true)
200  )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme023-0-market-bitcoin.clar](#)**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



653 lines (589 loc) · 24.6 KB

Code

Blame

Raw



```
1 ;; Title: BME023 Market bitcoin predictions
2 ;; Synopsis:
3 ;; Implements predicition markets for bitcoin users (see also bme023-0-market-predicting).
4 ;; Description:
5 ;; Provide binary and categorical prediction markets with
6 ;; bitcoin only transactions – no stx needed for gas. Works with
7 ;; clarity-bitcoin-lib-v5 for bitcoin catamaran swaps into markets.
8
9 (impl-trait .prediction-market-trait.prediction-market-trait)
10
11 (define-constant min-stake u100000) ;; Example: 100,000 satoshis (0.001 BTC)
12
13 ;; ----- CONSTANTS & TYPES -----
14 (define-constant MARKET_TYPE u3) ;; bitcoin tx market
15 (define-constant token 'SM3VDXK3WZZSA84XXFKAF15NNZX32CTSG82JFQ4.sbtc-token)
16
17 (define-constant RESOLUTION_OPEN u0)
18 (define-constant RESOLUTION_RESOLVING u1)
19 (define-constant RESOLUTION_DISPUTED u2)
20 (define-constant RESOLUTION_RESOLVED u3)
21
22 (define-constant err-unauthorised (err u10000))
23 (define-constant err-invalid-market-type (err u10001))
24 (define-constant err-amount-too-low (err u10002))
25 (define-constant err-wrong-market-type (err u10003))
26 (define-constant err-already-concluded (err u10004))
27 (define-constant err-market-not-found (err u10005))
28 (define-constant err-user-not-winner-or-claimed (err u10006))
29 (define-constant err-user-not-staked (err u10008))
30 (define-constant err-market-not-concluded (err u10009))
31 (define-constant err-insufficient-balance (err u10011))
32 (define-constant err-insufficient-contract-balance (err u10012))
33 (define-constant err-user-share-is-zero (err u10013))
34 (define-constant err-dao-fee-bips-is-zero (err u10014))
35 (define-constant err-disputer-must-have-stake (err u10015))
36 (define-constant err-dispute-window-elapsed (err u10016))
37 (define-constant err-market-not-resolving (err u10017))
38 (define-constant err-market-not-open (err u10018))
39 (define-constant err-dispute-window-not-elapsed (err u10019))
```

```

39 (define-constant err-dispute-window-not-stopped (err u10010))
40 (define-constant err-market-wrong-state (err u10020))
41 (define-constant err-invalid-token (err u10021))
42 (define-constant err-max-market-fee-bips-exceeded (err u10022))
43 (define-constant err-category-not-found (err u10023))
44 (define-constant err-too-few-categories (err u10024))
45 (define-constant err-element-expected (err u10025))
46 (define-constant err-winning-stake-not-zero (err u10026))
47 (define-constant err-losing-stake-is-zero (err u10027))
48 (define-constant err-transaction-segwit (err u10028))
49 (define-constant err-transaction-legacy (err u10029))
50 (define-constant err-transaction (err u1030))
51 (define-constant err-market-wallet (err u1031))
52 (define-constant err-transfer-forbidden (err u1032))
53
54 (define-data-var market-counter uint u0)
55 (define-data-var dispute-window-length uint u144)
56 (define-data-var dev-fee-bips uint u200)
57 (define-data-var dao-fee-bips uint u200)
58 (define-data-var market-fee-bips-max uint u1000)
59 (define-data-var market-create-fee uint u100000000)
60 (define-data-var dev-fund principal tx-sender)
61 (define-data-var resolution-agent principal tx-sender)
62 (define-data-var dao-treasury principal tx-sender)
63 (define-data-var creation-gated bool true)
64 (define-data-var market-wallet { version: (buff 1), hashbytes: (buff 32) } { version: 0x00, h
65 (define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)
66
67 ;; Data structure for each Market
68 ;; outcome: winning category
69 (define-map markets
70   uint
71   {
72       market-data-hash: (buff 32),
73       treasury: principal,
74       creator: principal,
75       market-fee-bips: uint,
76       resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
77       resolution-burn-height: uint,
78       categories: (list 10 (string-ascii 64)), ;; List of available categories
79       stakes: (list 10 uint), ;; Total staked per category
80       outcome: (optional uint),
81       concluded: bool
82   }
83 )
84
85 (define-map stake-balances
86   { market-id: uint, user: principal }
87   (list 10 uint)
88 )
89
90 ;; ----- access control -----
91 (define-public (is-dao-or-extension)
92   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
93 )
94
95 ;; ----- getters / setters -----

```



```
96
97 (define-public (set-dispute-window-length (length uint))
98   (begin
99     (try! (is-dao-or-extension))
100     (var-set dispute-window-length length)
101     (ok true)
102   )
103 )
104
105 (define-public (set-creation-gated (gated bool))
106   (begin
107     (try! (is-dao-or-extension))
108     (var-set creation-gated gated)
109     (ok true)
110   )
```



```

580         (market-fee-bips (get market-fee-bips md))
581         (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
582         (dao-fee (/ (* user-share (var-get dao-fee-bips)) u10000))
583         (market-fee (/ (* user-share market-fee-bips) u10000))
584         (user-share-net (- user-share (+ dao-fee market-fee)))
585     )
586 (begin
587     ;; Ensure inputs are valid
588     (asserts! (> winning-pool u0) err-amount-too-low)
589     (asserts! (> user-share-net u0) err-user-share-is-zero)
590     (asserts! (> dao-fee u0) err-dao-fee-bips-is-zero)
591
592     ;; Perform transfers
593     (as-contract
594         (begin
595             ;; Transfer user share, capped by initial contract balance
596
597             ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
598             ;;(try! (stx-transfer? dao-fee tx-sender (var-get dao-treasury)))
599
600             (if (> user-share-net u0)

```

```

601      (try! (contract-call? token transfer user-share-net tx-sender original-sender none)
602        true
603      )
604      (if (> daofee u0)
605        (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none)
606          true
607        )
608        (if (> marketfee u0)
609          (try! (contract-call? token transfer marketfee tx-sender treasury none))
610            true
611          )
612        )
613      )
614
615      ;; Zero out user stake
616      (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0))
617
618      ;; Log and return user share
619      (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
620      (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: tx-sender})
621      (ok user-share-net)
622    )
623  )
624 )
625
626 ;; the funds have arrived on bitcoin - so the sender here is the big market sbtc liquidity pool
627 (define-private (process-stake-transfer (amount uint))
628   (let (
629     ;;(sender-balance (stx-get-balance tx-sender))
630     (sender-balance (unwrap! (contract-call? token get-balance .bme023-0-market-bitcoin)
631       (fee (calculate-fee amount (var-get dev-fee-bips)))
632       (transfer-amount (- amount fee))
633     )
634     (begin
635       ;; Ensure amount is valid
636       (asserts! (>= amount u100) err-amount-too-low)
637       ;; Check tx-sender's balance
638       (asserts! (>= sender-balance amount) err-insufficient-balance)
639
640       ;; assume here the contract has the funds to cover payouts.
641       ;; in fact the liquidity will come from direct sbtc into this contract from the bitcoin
642       ;; (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-prec
643       (try! (as-contract (contract-call? token transfer fee .bme023-0-market-bitcoin (var-get
644
645         (ok transfer-amount)
646       )
647     )
648   )
649   (define-private (calculate-fee (amount uint) (fee-bips uint))
650     (let ((fee (/ (* amount fee-bips) u10000)))
651       fee
652     )
653   )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme023-0-market-predicting.clar](#)[Go to file](#)**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



495 lines (445 loc) · 19.5 KB

Code

Blame

Raw



```
1 ;; Title: BME023 Market Predicting
2 ;; Synopsis:
3 ;; Implements binary and categorical prediciton markets.
4 ;; Description:
5 ;; Market creation allows a new binary or categorical market to be set up.
6 ;; Off chain market data is verifiable via the markets data hash.
7 ;; Markets run in a specific token (stx, sbtc, bmg etc) the market is created
8 ;; with an allowed token. Allowed tokens are controlled by the DAO.
9 ;; Market creation can be gated via market proof and a market creator can
10 ;; set their own fee up to a max fee amount determined by the DAO.
11 ;; Anyone with the required token can stake as many times as they wish and for any choice
12 ;; of outcome. Resolution process begins via a call gated to the DAO controlled resolution ac
13 ;; address. The resolution can be challenged by anyone with a stake in the market
14 ;; If a challenge is made the dispute resolution process begins which requires a DAO vote
15 ;; to resolve - the outcome of the vote resolve the market and sets the outcome.
16 ;; If the dispute window passes without challenge or once the vote concludes the market is fu
17 ;; resolved and claims can then be made.
18
19 (use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
20 (impl-trait .prediction-market-trait.prediction-market-trait)
21
22 ;; ----- CONSTANTS & TYPES -----
23 ;; Market Types (1 => categorical market)
24 (define-constant MARKET_TYPE u1)
25
26 (define-constant RESOLUTION_OPEN u0)
27 (define-constant RESOLUTION_RESOLVING u1)
28 (define-constant RESOLUTION_DISPUTED u2)
29 (define-constant RESOLUTION_RESOLVED u3)
30
31 (define-constant err-unauthorised (err u10000))
32 (define-constant err-invalid-market-type (err u10001))
33 (define-constant err-amount-too-low (err u10002))
34 (define-constant err-wrong-market-type (err u10003))
35 (define-constant err-already-concluded (err u10004))
36 (define-constant err-market-not-found (err u10005))
37 (define-constant err-user-not-winner-or-claimed (err u10006))
38 (define-constant err-user-not-staked (err u10008))
39 (define-constant err-market-not-concluded (err u10009))
```

```

80 (define-constant err-market-not-concluded (err u10000))
81
82 (define-constant err-insufficient-balance (err u10011))
83
84 (define-constant err-insufficient-contract-balance (err u10012))
85
86 (define-constant err-user-share-is-zero (err u10013))
87
88 (define-constant err-dao-fee-bips-is-zero (err u10014))
89
90 (define-constant err-disputer-must-have-stake (err u10015))
91
92 (define-constant err-dispute-window-elapsed (err u10016))
93
94 (define-constant err-market-not-resolving (err u10017))
95
96 (define-constant err-market-not-open (err u10018))
97
98 (define-constant err-dispute-window-not-elapsed (err u10019))
99
100 (define-constant err-market-wrong-state (err u10020))
101
102 (define-constant err-invalid-token (err u10021))
103
104 (define-constant err-max-market-fee-bips-exceeded (err u10022))
105
106 (define-constant err-category-not-found (err u10023))
107
108 (define-constant err-too-few-categories (err u10024))
109
110 (define-constant err-element-expected (err u10025))
111
112 (define-constant err-winning-stake-not-zero (err u10026))
113
114 (define-constant err-losing-stake-is-zero (err u10027))
115
116
117 (define-data-var market-counter uint u0)
118 (define-data-var dispute-window-length uint u144)
119 (define-data-var dev-fee-bips uint u200)
120 (define-data-var dao-fee-bips uint u200)
121 (define-data-var market-fee-bips-max uint u1000)
122 (define-data-var market-create-fee uint u100000000)
123 (define-data-var dev-fund principal tx-sender)
124 (define-data-var resolution-agent principal tx-sender)
125 (define-data-var dao-treasury principal tx-sender)
126 (define-data-var creation-gated bool true)
127 (define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)
128
129 ;; Data structure for each Market
130 ;; outcome: winning category
131 (define-map markets
132   uint
133   {
134     market-data-hash: (buff 32),
135     token: principal,
136     treasury: principal,
137     creator: principal,
138     market-fee-bips: uint,
139     resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
140     resolution-burn-height: uint,
141     categories: (list 10 (string-ascii 64)), ;; List of available categories
142     stakes: (list 10 uint), ;; Total staked per category
143     outcome: (optional uint),
144     concluded: bool
145   }
146 )
147
148 (define-map stake-balances
149   { market-id: uint, user: principal }
150   (list 10 uint)
151 )
152 (define-map allowed-tokens principal bool)
153
154 ;; ----- access control -----

```

```
96      (define-public (is-dao-or-extension)
97        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
98          )
99
100      ;; ----- getters / setters -----
101      (define-public (set-allowed-token (token principal) (enabled bool))
102        (begin
103          (try! (is-dao-or-extension))
104          (print {event: "allowed-token", token: token, enabled: enabled})
105          (ok (map-set allowed-tokens token enabled))
106        )
107      )
108      (define-read-only (is-allowed-token (token principal))
109        (default-to false (map-get? allowed-tokens token))
110      )
```



```
422 (user-stake uint)
423 (winning-pool uint)
424 (total-pool uint)
425 (index-won uint) (token <ft-token>))
426 (let (
427     (md (unwrap! (map-get? markets market-id) err-market-not-found))
428     (original-sender tx-sender)
429     (treasury (get treasury md))
430     (market-fee-bips (get market-fee-bips md))
431     (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
432     (daofee (/ (* user-share (var-get dao-fee-bips)) u10000))
```

```

433         (marketfee (/ (* user-share market-fee-bips) u10000))
434         (user-share-net (- user-share (+ daofee marketfee)))
435     )
436 (begin
437     ;; Ensure inputs are valid
438     (asserts! (> winning-pool u0) err-amount-too-low)
439     (asserts! (> user-share-net u0) err-user-share-is-zero)
440
441     ;; Perform transfers
442     (as-contract
443         (begin
444             ;; Transfer user share, capped by initial contract balance
445             (if (> user-share-net u0)
446                 (try! (contract-call? token transfer user-share-net tx-sender original-sender none)
447                     true)
448                 )
449             (if (> daofee u0)
450                 (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none)
451                     true)
452                 )
453             (if (> marketfee u0)
454                 (try! (contract-call? token transfer marketfee tx-sender treasury none))
455                     true)
456             )
457         )
458     )
459
460     ;; Zero out user stake
461     (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0))
462
463     ;; Log and return user share
464     (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
465     (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: tx-sender}
466         (ok user-share-net))
467 )
468 )
469 )
470
471 (define-private (process-stake-transfer (amount uint) (token <ft-token>))
472     (let (
473         ;;(sender-balance (stx-get-balance tx-sender))
474         (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficient-balance))
475         (fee (calculate-fee amount (var-get dev-fee-bips)))
476         (transfer-amount (- amount fee))
477     )
478     (begin
479         ;; Ensure amount is valid
480         (asserts! (>= amount u100) err-amount-too-low)
481         ;; Check tx-sender's balance
482         (asserts! (>= sender-balance amount) err-insufficient-balance)
483
484         (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-predictor)
485             (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))
486             )
487         (ok transfer-amount)
488     )

```

```
489     )
490   )
491   (define-private (calculate-fee (amount uint) (fee-bips uint))
492     (let ((fee (/ (* amount fee-bips) u10000)))
493       fee
494     )
495   )
```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme023-0-market-scalar-dia.clar](#)

...

**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



594 lines (536 loc) · 23.4 KB

Code

Blame

Raw



```
1 ;; Title: BME023 Market scalar predictions
2 ;; Synopsis:
3 ;; Implements scalar predicition markets (see also bme023-0-market-predicting).
4 ;; Description:
5 ;; Scalar markets differ from binary/categorical markets (see bme023-0-market-predicting)
6 ;; in the type of categories and the mechanism for rsolution:
7 ;; Firstly, the categories are contiguous ranges of numbers with a min and max value. The wir
8 ;; category is decided by the range that the outcome selects. Secondly, scalar market outcome
9 ;; are determined by on-chain oracles. This contract uses the DIA oracle for selecting from
10 ;; possible outcomes.
11
12 (use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
13 (impl-trait .prediction-market-trait.prediction-market-trait)
14
15 ;; ----- CONSTANTS & TYPES -----
16 ;; Market Types (2 => range based markets)
17 (define-constant MARKET_TYPE u2)
18
19 ;; Price Feeds
20 ;; DIA_ORACLE 'SP1G48FZ4Y7JY8G2Z0N51QTCYGBQ6F4J43J77BQC0.dia-oracle
21 ;; DIA_ORACLE 'ST3Q982CNNQ00E3FH6853EMTA5FPF1M3ENJTHB8PY.dia-oracle
22 (define-constant STX_USD_FEED_ID 0xe62df6c8b4a85fe1a67db44dc12de5db330f7ac66b72dc658afedf0f4a
23
24 (define-constant DEFAULT_MARKET_DURATION u144) ;; ~1 day in Bitcoin blocks
25 (define-constant DEFAULT_COOL_DOWN_PERIOD u144) ;; ~1 day in Bitcoin blocks
26
27 (define-constant RESOLUTION_OPEN u0)
28 (define-constant RESOLUTION_RESOLVING u1)
29 (define-constant RESOLUTION_DISPUTED u2)
30 (define-constant RESOLUTION_RESOLVED u3)
31
32 (define-constant err-unauthorised (err u10000))
33 (define-constant err-invalid-market-type (err u10001))
34 (define-constant err-amount-too-low (err u10002))
35 (define-constant err-wrong-market-type (err u10003))
36 (define-constant err-already-concluded (err u10004))
37 (define-constant err-market-not-found (err u10005))
38 (define-constant err-user-not-winner-or-claimed (err u10006))
39 (define-constant err-user-not-staked (err u10008))
```

```

39 (define-constant err-user-not-staked (err u10000))
40 (define-constant err-market-not-concluded (err u10009))
41 (define-constant err-insufficient-balance (err u10011))
42 (define-constant err-insufficient-contract-balance (err u10012))
43 (define-constant err-user-share-is-zero (err u10013))
44 (define-constant err-dao-fee-bips-is-zero (err u10014))
45 (define-constant err-disputer-must-have-stake (err u10015))
46 (define-constant err-dispute-window-elapsed (err u10016))
47 (define-constant err-market-not-resolving (err u10017))
48 (define-constant err-market-not-open (err u10018))
49 (define-constant err-dispute-window-not-elapsed (err u10019))
50 (define-constant err-market-wrong-state (err u10020))
51 (define-constant err-invalid-token (err u10021))
52 (define-constant err-max-market-fee-bips-exceeded (err u10022))
53 (define-constant err-category-not-found (err u10023))
54 (define-constant err-too-few-categories (err u10024))
55 (define-constant err-element-expected (err u10025))
56 (define-constant err-winning-stake-not-zero (err u10026))
57 (define-constant err-losing-stake-is-zero (err u10027))
58 (define-constant err-unknown-stacks-block (err u10028))
59
60 (define-data-var market-counter uint u0)
61 (define-data-var dispute-window-length uint u144)
62 (define-data-var dev-fee-bips uint u200)
63 (define-data-var dao-fee-bips uint u200)
64 (define-data-var market-fee-bips-max uint u1000)
65 (define-data-var market-create-fee uint u100000000)
66 (define-data-var dev-fund principal tx-sender)
67 (define-data-var resolution-agent principal tx-sender)
68 (define-data-var dao-treasury principal tx-sender)
69 (define-data-var creation-gated bool true)
70 (define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)
71
72 ;; Data structure for each Market
73 ;; outcome: winning category
74 (define-map markets
75   uint
76   {
77       market-data-hash: (buff 32),
78       token: principal,
79       treasury: principal,
80       creator: principal,
81       market-fee-bips: uint,
82       resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
83       categories: (list 10 {min: uint, max: uint}), ;; Min (inclusive) and Max (exclusive)
84       stakes: (list 10 uint), ;; Total staked per category
85       outcome: (optional uint),
86       concluded: bool,
87       market-start: uint,
88       market-duration: uint,
89       cool-down-period: uint,
90       price-feed-id: (string-ascii 32), ;; DIA price feed ID (custom per market)
91       price-outcome: (optional uint)
92   }
93 )
94
95 (define-map stake-balances

```

```
96     { market-id: uint, user: principal }
97     (list 10 uint)
98 )
99 (define-map allowed-tokens principal bool)
100
101 ;; ----- access control -----
102 (define-public (is-dao-or-extension)
103     (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
104 )
105
106 ;; ----- getters / setters -----
107 (define-public (set-allowed-token (token principal) (enabled bool))
108     (begin
109         (try! (is-dao-or-extension))
110         (print {event: "allowed-token", token: token, enabled: enabled}))
```



```

521      (md (unwrap! (map-get? markets market-id) err-market-not-found))
522      (original-sender tx-sender)
523      (treasury (get treasury md))
524      (market-fee-bips (get market-fee-bips md))
525      (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
526      (dao-fee (/ (* user-share (var-get dao-fee-bips)) u10000))
527      (market-fee (/ (* user-share market-fee-bips) u10000))
528      (user-share-net (- user-share (+ dao-fee market-fee)))
529    )
530    (begin
531      ;; Ensure inputs are valid
532      (asserts! (> winning-pool u0) err-amount-too-low)
533      (asserts! (> user-share-net u0) err-user-share-is-zero)
534      (asserts! (> dao-fee u0) err-dao-fee-bips-is-zero)
535
536      ;; Perform transfers
537      (as-contract
538        (begin
539          ;; Transfer user share, capped by initial contract balance
540
541          ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
542          ;;(try! (stx-transfer? dao-fee tx-sender (var-get dao-treasury)))
543
544          (if (> user-share-net u0)

```

```

545         (try! (contract-call? token transfer user-share-net tx-sender original-sender nor
546             true
547         )
548         (if (> daofee u0)
549             (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none
550                 true
551             )
552             (if (> marketfee u0)
553                 (try! (contract-call? token transfer marketfee tx-sender treasury none))
554                 true
555             )
556         )
557     )
558
559     ;; Zero out user stake
560     (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0
561
562     ;; Log and return user share
563     (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
564     (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: t
565     (ok user-share-net)
566     )
567 )
568 )
569
570 (define-private (process-stake-transfer (amount uint) (token <ft-token>))
571     (let (
572         ;;(sender-balance (stx-get-balance tx-sender))
573         (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficien
574         (fee (calculate-fee amount (var-get dev-fee-bips)))
575         (transfer-amount (- amount fee))
576         )
577     (begin
578         ;; Ensure amount is valid
579         (asserts! (>= amount u100) err-amount-too-low)
580         ;; Check tx-sender's balance
581         (asserts! (>= sender-balance amount) err-insufficient-balance)
582
583         (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-scalar-
584         (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))
585
586         (ok transfer-amount)
587     )
588     )
589 )
590 (define-private (calculate-fee (amount uint) (fee-bips uint))
591     (let ((fee (/ (* amount fee-bips) u10000)))
592         fee
593     )
594 )

```

 [main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme023-0-market-scalar-pyth.clar](#)  Go to file**mijoco-btc** Inclusion of reputation soul bound sft.

32188d4 · last week



573 lines (517 loc) · 22.5 KB

Code

Blame

Raw



```
1 ;; Title: BME023-3 Market scalar predictions
2 ;; Synopsis:
3 ;; Implements scalar prediciton markets with pyth oracle resolution.
4 ;; Description:
5 ;; Scalar markets differ from binary/categorical markets (see bme023-0-market-predicting)
6 ;; in the type of categories and the mechanism for rsolution:
7 ;; Firstly, the categories are contiguous ranges of numbers with a min and max value. The wir
8 ;; category is decided by the range that the outcome selects. Secondly, scalar market outcome
9 ;; are determined by on-chain oracles.
10 ;; This contract uses the Pyth oracle for selecting from possible outcomes.
11
12 (use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
13 (impl-trait .prediction-market-trait.prediction-market-trait)
14
15 ;; ----- CONSTANTS & TYPES -----
16 ;; Market Types (2 => range based markets)
17 (define-constant MARKET_TYPE u2)
18
19 ;; Price Feeds
20 ;; PYTH_ORACLE 'ST20M5GABDT6WYJHXB5CDH4501V1Q65242SPRMXH.pyth-storage-v3
21 ;; PYTH_ORACLE 'SP3R4F6C1J3JQWWCVZ3S7FRRYPYMG6ZW6RZK31FXY.pyth-storage-v3
22
23 (define-constant DEFAULT_MARKET_DURATION u144) ;; ~1 day in Bitcoin blocks
24 (define-constant DEFAULT_COOL_DOWN_PERIOD u144) ;; ~1 day in Bitcoin blocks
25
26 (define-constant RESOLUTION_OPEN u0)
27 (define-constant RESOLUTION_RESOLVING u1)
28 (define-constant RESOLUTION_DISPUTED u2)
29 (define-constant RESOLUTION_RESOLVED u3)
30
31 (define-constant err-unauthorised (err u10000))
32 (define-constant err-invalid-market-type (err u10001))
33 (define-constant err-amount-too-low (err u10002))
34 (define-constant err-wrong-market-type (err u10003))
35 (define-constant err-already-concluded (err u10004))
36 (define-constant err-market-not-found (err u10005))
37 (define-constant err-user-not-winner-or-claimed (err u10006))
38 (define-constant err-user-not-staked (err u10008))
39 (define-constant err-market-not-concluded (err u10009))
```



```

39 (define-constant err-market-not-concluded (err u10000))
40 (define-constant err-insufficient-balance (err u10011))
41 (define-constant err-insufficient-contract-balance (err u10012))
42 (define-constant err-user-share-is-zero (err u10013))
43 (define-constant err-dao-fee-bips-is-zero (err u10014))
44 (define-constant err-disputer-must-have-stake (err u10015))
45 (define-constant err-dispute-window-elapsed (err u10016))
46 (define-constant err-market-not-resolving (err u10017))
47 (define-constant err-market-not-open (err u10018))
48 (define-constant err-dispute-window-not-elapsed (err u10019))
49 (define-constant err-market-wrong-state (err u10020))
50 (define-constant err-invalid-token (err u10021))
51 (define-constant err-max-market-fee-bips-exceeded (err u10022))
52 (define-constant err-category-not-found (err u10023))
53 (define-constant err-too-few-categories (err u10024))
54 (define-constant err-element-expected (err u10025))
55 (define-constant err-winning-stake-not-zero (err u10026))
56 (define-constant err-losing-stake-is-zero (err u10027))
57 (define-constant err-unknown-stacks-block (err u10028))
58
59 (define-data-var market-counter uint u0)
60 (define-data-var dispute-window-length uint u144)
61 (define-data-var dev-fee-bips uint u200)
62 (define-data-var dao-fee-bips uint u200)
63 (define-data-var market-fee-bips-max uint u1000)
64 (define-data-var market-create-fee uint u100000000)
65 (define-data-var dev-fund principal tx-sender)
66 (define-data-var resolution-agent principal tx-sender)
67 (define-data-var dao-treasury principal tx-sender)
68 (define-data-var creation-gated bool true)
69 (define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)
70
71 ;; Data structure for each Market
72 ;; outcome: winning category
73 (define-map markets
74   uint
75   {
76     market-data-hash: (buff 32),
77     token: principal,
78     treasury: principal,
79     creator: principal,
80     market-fee-bips: uint,
81     resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
82     categories: (list 10 {min: uint, max: uint}), ;; Min (inclusive) and Max (exclusive)
83     stakes: (list 10 uint), ;; Total staked per category
84     outcome: (optional uint),
85     concluded: bool,
86     market-start: uint,
87     market-duration: uint,
88     cool-down-period: uint,
89     price-feed-id: (buff 32), ;; Pyth price feed ID
90     price-outcome: (optional uint)
91   }
92 )
93
94 (define-map stake-balances
95   { market-id: uint, user: principal }

```

```
96     (list 10 uint)
97   )
98   (define-map allowed-tokens principal bool)
99
100  ;; ----- access control -----
101  (define-public (is-dao-or-extension)
102    (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
103  )
104
105  ;; ----- getters / setters -----
106  (define-public (set-allowed-token (token principal) (enabled bool))
107    (begin
108      (try! (is-dao-or-extension))
109      (print {event: "allowed-token", token: token, enabled: enabled})
110      (ok (map-set allowed-tokens token enabled))
```



```

500     (md (unwrap! (map-get? markets market-id) err-market-not-found))
501     (original-sender tx-sender)
502     (treasury (get treasury md))
503     (market-fee-bips (get market-fee-bips md))
504     (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
505     (dao-fee (/ (* user-share (var-get dao-fee-bips)) u10000))
506     (market-fee (/ (* user-share market-fee-bips) u10000))
507     (user-share-net (- user-share (+ dao-fee market-fee)))
508 )
509 (begin
510   ;; Ensure inputs are valid
511   (asserts! (> winning-pool u0) err-amount-too-low)
512   (asserts! (> user-share-net u0) err-user-share-is-zero)
513   (asserts! (> dao-fee u0) err-dao-fee-bips-is-zero)
514
515   ;; Perform transfers
516   (as-contract
517     (begin
518       ;; Transfer user share, capped by initial contract balance
519
520       ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
521       ;;(try! (stx-transfer? dao-fee tx-sender (var-get dao-treasury)))
522
523       (if (> user-share-net u0)
524         (try! (contract-call? token transfer user-share-net tx-sender original-sender none)
525           true
526         )
527         (if (> dao-fee u0)
528           (try! (contract-call? token transfer dao-fee tx-sender (var-get dao-treasury) none)
529             true
530           )
531           (if (> market-fee u0)
532             (try! (contract-call? token transfer market-fee tx-sender treasury none))
533             true
534           )
535         )
536       )
537
538   ;; Zero out user stake
539   (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0))
540
541   ;; Log and return user share
542   (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
543   (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: tx-sender})
544   (ok user-share-net)

```

```

545     )
546   )
547 )
548
549 (define-private (process-stake-transfer (amount uint) (token <ft-token>))
550   (let (
551     ;;(sender-balance (stx-get-balance tx-sender))
552     (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficient-
553     (fee (calculate-fee amount (var-get dev-fee-bips)))
554     (transfer-amount (- amount fee))
555   )
556   (begin
557     ;; Ensure amount is valid
558     (asserts! (>= amount u100) err-amount-too-low)
559     ;; Check tx-sender's balance
560     (asserts! (>= sender-balance amount) err-insufficient-balance)
561
562     (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-scalar-
563     (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))
564
565     (ok transfer-amount)
566   )
567 )
568 )
569 (define-private (calculate-fee (amount uint) (fee-bips uint))
570   (let ((fee (/ (* amount fee-bips) u10000)))
571     fee
572   )
573 )

```

[main](#) [bigmarket-dao / contracts / extensions](#)
[/ bme030-0-reputation-token.clar](#)

...

 **mijoco-btc** Release 2. Changes include new soul bound sft based reputation system9f68c60 · 5 days ago 

293 lines (253 loc) · 10.4 KB

[Code](#) [Blame](#)

Raw



```
1 ;; Title: BME030 Reputation Token
2 ;; Synopsis:
3 ;; Wraps reputation scheme within a non-transferable soulbound semi fungible token (see sip-0
4 ;; Description:
5 ;; The reputation token is a SIP-013 compliant token that is controlled by active DAO extensi
6 ;; It facilitates hierarchical reputation and rewards based on engagements across a number of
7 ;; BigMarket DAO features and use cases.
8
9 (impl-trait 'SPDBEG5X8XD50SPM1JJH0E5CTXGDV5NJTAKAKR5V.sip013-semi-fungible-token-trait.sip013
10 (impl-trait 'SPDBEG5X8XD50SPM1JJH0E5CTXGDV5NJTAKAKR5V.sip013-transfer-many-trait.sip013-trans
11
12 (define-constant err-unauthorised (err u30001))
13 (define-constant err-already-minted (err u30002))
14 (define-constant err-soulbound (err u30003))
15 (define-constant err-insufficient-balance (err u30004))
16 (define-constant err-zero-amount (err u30005))
17 (define-constant err-claims-old-epoch (err u30006))
18 (define-constant err-claims-zero-rep (err u30007))
19 (define-constant err-claims-zero-total (err u30008))
20 (define-constant err-invalid-tier (err u30009))
21
22 (define-constant max-tier u20)
23
24 (define-data-var token-name (string-ascii 32) "BigMarket Reputation Token")
25 (define-data-var token-symbol (string-ascii 10) "BIGR")
26
27 (define-fungible-token bigr-token)
28 (define-non-fungible-token bigr-id { token-id: uint, owner: principal })
29
30 (define-map balances { token-id: uint, owner: principal } uint)
31 (define-map supplies uint uint)
32 (define-map last-claimed-epoch { who: principal } uint)
33 (define-map tier-weights uint uint)
34
35 (define-data-var reward-per-epoch uint u1000000000) ;; 1000 BIG (in micro units)
36 (define-data-var overall-supply uint u0)
37
```

```

38 ;; -----
39 ;; DAO Control Check
40 ;; -----
41 (define-public (is-dao-or-extension)
42   (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
43   )
44
45   ;; -----
46   ;; Trait Implementations
47   ;; -----
48   (define-read-only (get-balance (token-id uint) (who principal))
49     (ok (default-to u0 (map-get? balances { token-id: token-id, owner: who })))
50   )
51
52   (define-read-only (get-symbol)
53     (ok (var-get token-symbol))
54   )
55
56   (define-read-only (get-name)
57     (ok (var-get token-name))
58   )
59
60   (define-read-only (get-overall-balance (who principal))
61     (ok (ft-get-balance bigr-token who))
62   )
63
64   (define-read-only (get-total-supply (token-id uint))
65     (ok (default-to u0 (map-get? supplies token-id)))
66   )
67
68   (define-read-only (get-overall-supply)
69     (ok (var-get overall-supply))
70   )
71
72   (define-read-only (get-decimals (token-id uint)) (ok u0))
73
74   (define-read-only (get-token-uri (token-id uint))
75     (ok none)
76   )
77
78   (define-public (set-reward-per-epoch (new-reward uint))
79     (begin
80       (try! (is-dao-or-extension))
81       (var-set reward-per-epoch new-reward)
82       (ok true)
83     )
84   )
85   (define-public (set-tier-weight (token-id uint) (weight uint))
86     (begin
87       (try! (is-dao-or-extension))
88       (map-set tier-weights token-id weight)
89       (ok true)
90     )
91   )
92
93   ;; -----

```

```
94 ;; Mint / Burn
95 ;; -----
96 (define-public (mint (recipient principal) (token-id uint) (amount uint))
97   (begin
98     (try! (is-dao-or-extension))
99     (asserts! (> amount u0) err-zero-amount)
100    (asserts! (and (> token-id u0) (<= token-id max-tier)) err-invalid-tier)
101    (try! (ft-mint? bgr-token amount recipient))
102    (try! (tag-nft { token-id: token-id, owner: recipient }))
103    (map-set balances { token-id: token-id, owner: recipient }
104      (+ amount (default-to u0 (map-get? balances { token-id: token-id, owner: recipient }))))
105    (map-set supplies token-id (+ amount (default-to u0 (map-get? supplies token-id))))
106    (var-set overall-supply (+ (var-get overall-supply) amount))
107    (print { event: "sft_mint", token-id: token-id, amount: amount, recipient: recipient })
108    (ok true)
109  )
110 )
```



```

220         )
221     )
222     (ok true)
223 )
224 )
225 )
226
227 (define-read-only (get-epoch)
228     (/ burn-block-height u4000)
229 )
230
231 (define-read-only (get-last-claimed-epoch)
232     (default-to u0 (map-get? last-claimed-epoch { who: tx-sender })))
233 )
234
235 ;; -----
236 ;; Helpers
237 ;; -----
238 (define-private (tag-nft (nft-token-id { token-id: uint, owner: principal }))
239     (begin
240         (if (is-some (nft-get-owner? bigr-id nft-token-id))
241             (try! (nft-burn? bigr-id nft-token-id (get owner nft-token-id)))
242             true)
243         (nft-mint? bigr-id nft-token-id (get owner nft-token-id))
244     )
245 )
246
247 (define-private (transfer-many-iter (item { token-id: uint, amount: uint, sender: principal,
248     (match prev ok-prev (transfer (get token-id item) (get amount item) (get sender item) (get
249 )
250
251 (define-private (transfer-many-memo-iter (item { token-id: uint, amount: uint, sender: princi
252     (match prev ok-prev (transfer-memo (get token-id item) (get amount item) (get sender item)
253 )
254
255 ;; dynamic weighted totals for user
256 (define-read-only (get-weighted-rep (user principal))
257     (let (
258         (tiers (list u1 u2 u3 u4 u5 u6 u7 u8 u9 u10))
259         (result (fold add-weighted-rep-for-user tiers (tuple (acc u0) (user user)))))
260     )
261     (ok (get acc result))
262 )

```

```

263 )
264
265 (define-private (add-weighted-rep-for-user (token-id uint) (state (tuple (acc uint) (user pri
266   (let (
267     (acc (get acc state))
268     (user (get user state))
269     (bal-at-tier (default-to u0 (map-get? balances {token-id: token-id, owner: user})))
270     (weight-at-tier (default-to u1 (map-get? tier-weights token-id)))
271   )
272   (tuple (acc (+ acc (* bal-at-tier weight-at-tier))) (user user))
273 )
274 )
275
276 ;; dynamic weighted totals for overall supply pool
277 (define-read-only (get-weighted-supply)
278   (let (
279     (tiers (list u1 u2 u3 u4 u5 u6 u7 u8 u9 u10))
280     (result (fold add-weighted-supply-for-tier tiers u0))
281   )
282   (ok result)
283 )
284 )
285
286 (define-private (add-weighted-supply-for-tier (token-id uint) (acc uint))
287   (let (
288     (tier-supply (default-to u0 (map-get? supplies token-id)))
289     (weight (default-to u1 (map-get? tier-weights token-id)))
290   )
291   (+ acc (* tier-supply weight))
292 )
293 )

```