radicleart opinion poll contract 8642fd0 · last week

Executable File · 87 lines (70 loc) · 2.69 KB

Code Blame Raw

```
;; Bitcoin DAO
;; Author Mike Cohen - based on Marvin Janssen' Executor DAO
;; Synopsis:
;; bitcoin-dao is the core of the dao framework.
;; Description:
;; Valid extensions must be registered here. The DAO is bootstrapped
;; by calling construct with a bootstrap proposal.

(use-trait extension-trait .extension-trait.extension-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)

(define-constant err-unauthorised (err u1000))
(define-constant err-already-executed (err u1001))
(define-constant err-invalid-extension (err u1002))

(define-data-var executive principal tx-sender)
(define-map executed-proposals principal uint)
(define-map extensions principal bool)

;; --- Authorisation check

(define-private (is-self-or-extension)
        (ok (asserts! (or (is-eq tx-sender (as-contract tx-sender)) (is-extension contract-ca
)

;; --- Extensions

(define-read-only (is-extension (extension principal))
        (default-to false (map-get? extensions extension))
)

(define-public (set-extension (extension principal) (enabled bool))
        (begin
                (try! (is-self-or-extension))
                (print {event: "extension", extension: extension, enabled: enabled})
                (ok (map-set extensions extension enabled))
        )
)

```

```
(define-private (set-extensions-iter (item {extension: principal, enabled: bool}))
        (begin
                (print {event: "extension", extension: (get extension item), enabled: (get er
                (map-set extensions (get extension item) (get enabled item))
        )
)

(define-public (set-extensions (extension-list (list 200 {extension: principal, enabled: bool
        (begin
                (try! (is-self-or-extension))
                (ok (map set-extensions-iter extension-list))
        )
)

;; --- Proposals

(define-read-only (executed-at (proposal <proposal-trait>))
        (map-get? executed-proposals (contract-of proposal))
)

(define-public (execute (proposal <proposal-trait>) (sender principal))
        (begin
                (try! (is-self-or-extension))
                (asserts! (map-insert executed-proposals (contract-of proposal) stacks-block-
                (print {event: "execute", proposal: proposal})
                (as-contract (contract-call? proposal execute sender))
        )
)

;; --- Bootstrap

(define-public (construct (proposal <proposal-trait>))
        (let ((sender tx-sender))
                (asserts! (is-eq sender (var-get executive)) err-unauthorised)
                (var-set executive (as-contract tx-sender))
                (as-contract (execute proposal sender))
        )
)

;; --- Extension requests

(define-public (request-extension-callback (extension <extension-trait>) (memo (buff 34)))
        (let ((sender tx-sender))
                (asserts! (is-extension contract-caller) err-invalid-extension)
                (asserts! (is-eq contract-caller (contract-of extension)) err-invalid-extensi
                (as-contract (contract-call? extension callback sender memo))
        )
)
```

radicleart first commit 133764b · 4 hours ago

Executable File · 168 lines (132 loc) · 4.14 KB

Code Blame Raw

```
;; Title: BDE000 Governance Token
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On:
;; Synopsis:
;; This extension defines the governance token of Bitcoin DAO.
;; Description:
;; The governance token is a simple SIP010-compliant fungible token
;; with some added functions to make it easier to manage by
;; Bitcoin DAO proposals and extensions.

(impl-trait .governance-token-trait.governance-token-trait)
(impl-trait .sip010-ft-trait.sip010-ft-trait)
(impl-trait .extension-trait.extension-trait)

(define-constant err-unauthorised (err u3000))
(define-constant err-not-token-owner (err u4))

(define-fungible-token bdg-token)
(define-fungible-token bdg-token-locked)

(define-data-var token-name (string-ascii 32) "{{token_name}}")
(define-data-var token-symbol (string-ascii 10) "{{symbol}}")
(define-data-var token-uri (optional (string-utf8 256)) (some u"{{token_uri}}"))
(define-data-var token-decimals uint u6)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; --- Internal DAO functions

;; governance-token-trait

(define-public (bdg-transfer (amount uint) (sender principal) (recipient principal))
        (begin
                (try! (is-dao-or-extension))
                (ft-transfer? bdg-token amount sender recipient)
```

```
(try! (ft-transfer? bdg-token amount sender recipient))
        )
    )

    (define-public (bdg-lock (amount uint) (owner principal))
        (begin
            (try! (is-dao-or-extension))
            (try! (ft-burn? bdg-token amount owner))
            (ft-mint? bdg-token-locked amount owner)
        )
    )

    (define-public (bdg-unlock (amount uint) (owner principal))
        (begin
            (try! (is-dao-or-extension))
            (try! (ft-burn? bdg-token-locked amount owner))
            (ft-mint? bdg-token amount owner)
        )
    )

    (define-public (bdg-mint (amount uint) (recipient principal))
        (begin
            (try! (is-dao-or-extension))
            (ft-mint? bdg-token amount recipient)
        )
    )

    (define-public (bdg-burn (amount uint) (owner principal))
        (begin
            (try! (is-dao-or-extension))
            (ft-burn? bdg-token amount owner)

        )
    )

    ;; Other

    (define-public (set-name (new-name (string-ascii 32)))
        (begin
            (try! (is-dao-or-extension))
            (ok (var-set token-name new-name))
        )
    )

    (define-public (set-symbol (new-symbol (string-ascii 10)))
        (begin
            (try! (is-dao-or-extension))
            (ok (var-set token-symbol new-symbol))
        )
    )

    (define-public (set-decimals (new-decimals uint))
        (begin
            (try! (is-dao-or-extension))
            (ok (var-set token-decimals new-decimals))
        )
    )
```

```
96
97      (define-public (set-token-uri (new-uri (optional (string-utf8 256))))
98              (begin
99                      (try! (is-dao-or-extension))
100                     (ok (var-set token-uri new-uri))
101             )
102     )
103
104     (define-private (bdg-mint-many-iter (item {amount: uint, recipient: principal}))
105             (ft-mint? bdg-token (get amount item) (get recipient item))
106     )
107
108     (define-public (bdg-mint-many (recipients (list 200 {amount: uint, recipient: principal})))
109             (begin
110                     (try! (is-dao-or-extension))
111                     (ok (map bdg-mint-many-iter recipients))
112             )
113     )
114
115     ;; --- Public functions
116
117     ;; sip010-ft-trait
118
119     (define-public (transfer (amount uint) (sender principal) (recipient principal) (memo (optior
120             (begin
121                     (asserts! (or (is-eq tx-sender sender) (is-eq contract-caller sender)) err-no
122                     (ft-transfer? bdg-token amount sender recipient)
123             )
124     )
125
126     (define-read-only (get-name)
127             (ok (var-get token-name))
128     )
129
130     (define-read-only (get-symbol)
131             (ok (var-get token-symbol))
132     )
133
134     (define-read-only (get-decimals)
135             (ok (var-get token-decimals))
136     )
137
138     (define-read-only (get-balance (who principal))
139             (ok (+ (ft-get-balance bdg-token who) (ft-get-balance bdg-token-locked who)))
140     )
141
142     (define-read-only (get-total-supply)
143             (ok (+ (ft-get-supply bdg-token) (ft-get-supply bdg-token-locked)))
144     )
145
146     (define-read-only (get-token-uri)
147             (ok (var-get token-uri))
148     )
149
150     ;; governance-token-trait
151
```

```
152     (define-read-only (bdg-get-balance (who principal))
153             (get-balance who)
154     )
155
156     (define-read-only (bdg-has-percentage-balance (who principal) (factor uint))
157             (ok (>= (* (unwrap-panic (get-balance who)) factor) (* (unwrap-panic (get-total-suppl
158     )
159
160     (define-read-only (bdg-get-locked (owner principal))
161             (ok (ft-get-balance bdg-token-locked owner))
162     )
163
164     ;; --- Extension callback
165
166     (define-public (callback (sender principal) (memo (buff 34)))
167             (ok true)
168     )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 152 lines (130 loc) · 5.57 KB

Code Blame Raw

```
;; Title: BDE001 Snapshot Proposal Voting
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On:
;; Synopsis:
;; This extension is a concept that allows all STX holders to
;; vote on proposals based on their STX balance.
;; Description:
;; This extension allows anyone with STX to vote on proposals. The maximum upper
;; bound, or voting power, depends on the amount of STX tokens the tx-sender
;; owned at the start block height of the proposal. The name "snapshot" comes
;; from the fact that the extension effectively uses the STX balance sheet
;; at a specific block heights to determine voting power.
;; Custom majority thresholds for voting are also possible on a per proposal basis.
;; A custom majority of 66% mean the percent of votes for must be greater than 66 for
;; the vote to carry.

(impl-trait .extension-trait.extension-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)

(define-constant err-unauthorised (err u3000))
(define-constant err-proposal-already-executed (err u3001))
(define-constant err-proposal-already-exists (err u3002))
(define-constant err-unknown-proposal (err u3003))
(define-constant err-proposal-already-concluded (err u3004))
(define-constant err-proposal-inactive (err u3005))
(define-constant err-insufficient-voting-capacity (err u3006))
(define-constant err-end-burn-height-not-reached (err u3007))
(define-constant err-not-majority (err u3008))
(define-constant err-exceeds-voting-cap (err u3009))

(define-constant custom-majority-upper u10000)
(define-constant vote-cap u140000000000)

(define-map proposals
        principal
        {
                votes-for: uint,
                votes-against: uint,
                start-height-stacks: uint,
```

```
                start-burn-height: uint,
                end-burn-height: uint,
                concluded: bool,
                passed: bool,
                custom-majority: (optional uint), ;; u10000 = 100%
                proposer: principal
        }
)

(define-map member-total-votes {proposal: principal, voter: principal} uint)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; --- Internal DAO functions

;; Proposals

(define-public (add-proposal (proposal <proposal-trait>) (data {start-height-stacks: uint, st
        (begin
                (try! (is-dao-or-extension))
                (asserts! (is-none (contract-call? .bitcoin-dao executed-at proposal)) err-pr
                (asserts! (match (get custom-majority data) majority (> majority u5000) true)
                (print {event: "propose", proposal: proposal, proposer: tx-sender})
                (ok (asserts! (map-insert proposals (contract-of proposal) (merge {votes-for:
        )
)

;; --- Public functions

;; Proposals

(define-read-only (get-proposal-data (proposal principal))
        (map-get? proposals proposal)
)

;; Votes

(define-read-only (get-current-total-votes (proposal principal) (voter principal))
        (default-to u0 (map-get? member-total-votes {proposal: proposal, voter: voter}))
)

(define-read-only (get-historical-values (height uint) (who principal))
        (at-block (unwrap! (get-stacks-block-info? id-header-hash height) none)
                (some
                        {
                                user-balance: (stx-get-balance who),
                                voting-cap: vote-cap,
                                ;;voting-cap: (contract-call? 'SP000000000000000000002Q6VF78.
                        }
                )
        )
)
```

```
96
97      (define-public (vote (amount uint) (for bool) (proposal principal))
98              (let
99                      (
100                             (proposal-data (unwrap! (map-get? proposals proposal) err-unknown-pro
101                             (new-total-votes (+ (get-current-total-votes proposal tx-sender) amou
102                             (historical-values (unwrap! (get-historical-values (get start-height-
103                     )
104                     (asserts! (>= burn-block-height (get start-burn-height proposal-data)) err-pr
105                     (asserts! (< burn-block-height (get end-burn-height proposal-data)) err-propo
106                     (asserts!
107                             (<= new-total-votes (get user-balance historical-values))
108                             err-insufficient-voting-capacity
109                     )
110                     (asserts!
111                             (< new-total-votes (get voting-cap historical-values))
112                             err-exceeds-voting-cap)
113
114                     (map-set member-total-votes {proposal: proposal, voter: tx-sender} new-total-
115                     (map-set proposals proposal
116                             (if for
117                                     (merge proposal-data {votes-for: (+ (get votes-for proposal-d
118                                     (merge proposal-data {votes-against: (+ (get votes-against pr
119                             )
120                     )
121                     (print {event: "vote", proposal: proposal, voter: tx-sender, for: for, amount
122                     (ok true)
123             )
124     )
125
126     ;; Conclusion
127
128     (define-public (conclude (proposal <proposal-trait>))
129             (let
130                     (
131                             (proposal-data (unwrap! (map-get? proposals (contract-of proposal)) e
132                             (passed
133                                     (match (get custom-majority proposal-data)
134                                             majority (> (* (get votes-for proposal-data) custom-m
135                                             (> (get votes-for proposal-data) (get votes-against p
136                                     )
137                             )
138                     )
139                     (asserts! (not (get concluded proposal-data)) err-proposal-already-concluded)
140                     (asserts! (>= burn-block-height (get end-burn-height proposal-data)) err-end-
141                     (map-set proposals (contract-of proposal) (merge proposal-data {concluded: tr
142                     (print {event: "conclude", proposal: proposal, passed: passed})
143                     (and passed (try! (contract-call? .bitcoin-dao execute proposal tx-sender)))
144                     (ok passed)
145             )
146     )
147
148     ;; --- Extension callback
149
150     (define-public (callback (sender principal) (memo (buff 34)))
151             (ok true)
```

```
152         )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 285 lines (256 loc) · 11.2 KB

Code Blame Raw

```
;; Title: BDE001 Snapshot Proposal Voting
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On:
;; Synopsis:
;; This extension is a concept that allows all STX holders to
;; vote on proposals based on their STX balance.
;; Description:
;; This extension allows anyone with STX to vote on proposals. The maximum upper
;; bound, or voting power, depends on the amount of STX tokens the tx-sender
;; owned at the start block height of the proposal. The name "snapshot" comes
;; from the fact that the extension effectively uses the STX balance sheet
;; at a specific block heights to determine voting power.
;; Custom majority thresholds for voting are also possible on a per proposal basis.
;; A custom majority of 66% mean the percent of votes for must be greater than 66 for
;; the vote to carry.

(impl-trait .extension-trait.extension-trait)
(impl-trait .voting-trait.voting-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)

(define-constant err-unauthorised (err u3000))
(define-constant err-proposal-already-executed (err u3001))
(define-constant err-proposal-already-exists (err u3002))
(define-constant err-unknown-proposal (err u3003))
(define-constant err-proposal-already-concluded (err u3004))
(define-constant err-proposal-inactive (err u3005))
(define-constant err-insufficient-voting-capacity (err u3006))
(define-constant err-end-burn-height-not-reached (err u3007))
(define-constant err-not-majority (err u3008))
(define-constant err-proposal-start-no-reached (err u3009))
(define-constant err-historical-data (err u3010))

(define-constant custom-majority-upper u10000)

(define-constant structured-data-prefix 0x534950303138)
(define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
        {
                name: "BigMarket",
                version: "1.0.0",
```

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

```
96    )
97
98    (define-read-only (get-historical-values (height uint) (who principal))
99      (at-block (unwrap! (get-stacks-block-info? id-header-hash height) none)
100       (let (
101         (account-data (stx-account who)) ;; Fetch the STX account data
102       )
103         (some
104           {
105             user-balance: (tuple
106               (unlocked (get unlocked account-data))
107               (locked (get locked account-data))
108             )
109           }
110         )
```

```
212                             (map-set proposals proposal
213                                     (if for
214                                             (merge proposal-data {votes-for: (+ (get votes-for pr
215                                             (merge proposal-data {votes-against: (+ (get votes-ag
216                                     )
217                             )
218                             (map-set member-total-votes {proposal: proposal, voter: voter} new-to
219                             (map-set voter-timestamps {proposal: proposal, voter: voter} timestam
220                             (print {event: "vote", sip18: true,  proposal: proposal, voter: voter
221                             (ok u1) ;; Vote processed successfully
222                 )
223               )
224                   (begin
225                 (ok u0) ;; Invalid signature, skip vote
226                   )
227             )
228           )
229         )
230
231         (define-read-only (verify-signature (hash (buff 32)) (signature (buff 65)) (signer principal)
232                 (is-eq (principal-of? (unwrap! (secp256k1-recover? hash signature) false)) (ok signer
233         )
234
235         (define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
236                 (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
237         )
238
239         (define-read-only (verify-signed-tuple
240             (message-data (tuple
241                             (attestation (string-ascii 100))
242                             (proposal principal)
243                             (timestamp uint)
244                             (vote bool)
245                             (voter principal)
246                             (voting_power uint)))
247             (signature (buff 65))
248             (signer principal))
249           (let
250             (
251               ;; Compute the structured data hash
252                 (structured-data-hash (sha256 (unwrap! (to-consensus-buff? message-data) err-unauthor
253             )
254             ;; Verify the signature using the computed hash
255             (ok (verify-signed-structured-data structured-data-hash signature signer))
256           )
257         )
258
259         ;; Conclusion
260
261         (define-public (conclude (proposal <proposal-trait>))
262                 (let
263                         (
```

```
264                         (proposal-data (unwrap! (map-get? proposals (contract-of proposal)) e
265                         (passed
266                                 (match (get custom-majority proposal-data)
267                                         majority (> (* (get votes-for proposal-data) custom-m
268                                         (> (get votes-for proposal-data) (get votes-against p
269                                 )
270                         )
271                 )
272                 (asserts! (not (get concluded proposal-data)) err-proposal-already-concluded)
273                 (asserts! (>= burn-block-height (get end-burn-height proposal-data)) err-end-
274                 (map-set proposals (contract-of proposal) (merge proposal-data {concluded: tr
275                 (print {event: "conclude", proposal: proposal, passed: passed})
276                 (and passed (try! (contract-call? .bitcoin-dao execute proposal tx-sender)))
277                 (ok passed)
278         )
279 )
280
281 ;; --- Extension callback
282
283 (define-public (callback (sender principal) (memo (buff 34)))
284         (ok true)
285 )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 155 lines (128 loc) · 6.05 KB

Code Blame Raw

```
;; Title: BDE002 Funded Custom End Proposal Submission
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On: BDE001, BDE007
;; Synopsis:
;; This extension part of the core of Bitcoin DAO. It allows members to
;; bring proposals to the voting phase by funding them with a preset amount
;; of tokens.
;; Description:
;; The level of funding is determined by a DAO parameter and can be changed by proposal.
;; Any funder can reclaim their stx up to the point the proposal is fully funded and submitte
;; Proposals can also be marked as refundable in which case a funder can reclaim their stx
;; even after submission (during or after the voting period).
;; This extension provides the ability for the final funding transaction to set a
;; custom majority for voting. This changes the threshold from the
;; default of 50% to anything up to 100%.

(impl-trait .extension-trait.extension-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)
(use-trait voting-trait .voting-trait.voting-trait)

(define-constant err-unauthorised (err u3100))
(define-constant err-not-governance-token (err u3101))
(define-constant err-insufficient-balance (err u3102))
(define-constant err-unknown-parameter (err u3103))
(define-constant err-proposal-minimum-start-delay (err u3104))
(define-constant err-proposal-minimum-duration (err u3105))
(define-constant err-already-funded (err u3106))
(define-constant err-nothing-to-refund (err u3107))
(define-constant err-refund-not-allowed (err u3108))

(define-map refundable-proposals principal bool)
(define-map funded-proposals principal bool)
(define-map proposal-funding principal uint)
(define-map funding-per-principal {proposal: principal, funder: principal} uint)

(define-map parameters (string-ascii 30) uint)

(map-set parameters "funding-cost" u500000) ;; funding cost in uSTX. 5 STX in this case.
(map-set parameters "minimum-proposal-start-delay" u6) ;; eg 6 = ~1 hour minimum delay before
```

```
(map-set parameters "minimum-proposal-duration" u72) ;; eg 72 = ~1/2 days minimum duration of

;; --- Authorisation check

(define-public (is-dao-or-extension)
	(ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; --- Internal DAO functions

;; Proposals

(define-private (submit-proposal-for-vote (voting-contract <voting-trait>) (proposal <proposa
	(contract-call? voting-contract add-proposal
		proposal
		{
			start-height-stacks: start-height-stacks,
			start-burn-height: start-burn-height,
			end-burn-height: (+ start-burn-height duration),
			custom-majority: custom-majority,
			proposer: tx-sender ;; change to original submitter
		}
	)
)

;; Parameters

(define-public (set-parameter (parameter (string-ascii 30)) (value uint))
	(begin
		(try! (is-dao-or-extension))
		(try! (get-parameter parameter))
		(ok (map-set parameters parameter value))
	)
)

;; Refunds

(define-public (set-refundable (proposal principal) (refundable bool))
	(begin
		(try! (is-dao-or-extension))
		(ok (map-set refundable-proposals proposal refundable))
	)
)

;; --- Public functions

;; Parameters

(define-read-only (get-parameter (parameter (string-ascii 30)))
	(ok (unwrap! (map-get? parameters parameter) err-unknown-parameter))
)

;; Funding status

(define-read-only (is-proposal-funded (proposal principal))
	(default-to false (map-get? funded-proposals proposal))
```

```
)

(define-read-only (get-proposal-funding (proposal principal))
        (default-to u0 (map-get? proposal-funding proposal))
)

(define-read-only (get-proposal-funding-by-principal (proposal principal) (funder principal))
        (default-to u0 (map-get? funding-per-principal {proposal: proposal, funder: funder}))
)

(define-read-only (can-refund (proposal principal) (funder principal))
        (or
                (default-to false (map-get? refundable-proposals proposal))
                (and (not (is-proposal-funded proposal)) (is-eq funder tx-sender))
        )
)

;; Proposals

(define-public (fund (voting-contract <voting-trait>) (proposal <proposal-trait>) (start-dela
        (let
                (
                        (proposal-principal (contract-of proposal))
                        (current-total-funding (get-proposal-funding proposal-principal))
                        (funding-cost (try! (get-parameter "funding-cost")))
                        (difference (if (> funding-cost current-total-funding) (- funding-cos
                        (funded (<= difference amount))
                        (transfer-amount (if funded difference amount))
                )
                (asserts! (not (is-proposal-funded proposal-principal)) err-already-funded)
                (and (> transfer-amount u0) (try! (stx-transfer? transfer-amount tx-sender .b
                (map-set funding-per-principal {proposal: proposal-principal, funder: tx-send
                (map-set proposal-funding proposal-principal (+ current-total-funding transfe
                (asserts! funded (ok false))
                (asserts! (>= start-delay (try! (get-parameter "minimum-proposal-start-delay"
                (asserts! (>= duration (try! (get-parameter "minimum-proposal-duration"))) er
                (map-set funded-proposals proposal-principal true)
                (submit-proposal-for-vote voting-contract proposal stacks-block-height (+ bur
        )
)

(define-public (refund (proposal principal) (funder (optional principal)))
        (let
                (
                        (recipient (default-to tx-sender funder))
                        (refund-amount (get-proposal-funding-by-principal proposal recipient)
                )
                (asserts! (> refund-amount u0) err-nothing-to-refund)
                (asserts! (can-refund proposal recipient) err-refund-not-allowed)
                (map-set funding-per-principal {proposal: proposal, funder: recipient} u0)
                (map-set proposal-funding proposal (- (get-proposal-funding proposal) refund-
                (contract-call? .bde006-treasury stx-transfer refund-amount recipient none)
        )
)

;; --- Extension callback
```

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

radicleart first commit 133764b · 4 hours ago

Executable File · 80 lines (66 loc) · 2.89 KB

Code Blame Raw

```
;; Title: BDE003 Core Proposals
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On: BDE001
;; Synopsis:
;; This extension allows for the creation of core proposals by a few trusted
;; principals.
;; Description:
;; Only a list of trusted principals, designated as the
;; "core team", can create core proposals. The core proposal
;; extension has an optional ~3 month sunset period, after which no more core
;; proposals can be made - set it to 0 to disable. The core team members, sunset period, and
;; core vote duration can be changed by means of a future proposal.

(impl-trait .extension-trait.extension-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)
(use-trait voting-trait .voting-trait.voting-trait)

(define-data-var core-team-sunset-height uint u0) ;; does not expire by default - can be char

(define-constant err-unauthorised (err u3300))
(define-constant err-not-core-team-member (err u3301))
(define-constant err-sunset-height-reached (err u3302))
(define-constant err-sunset-height-in-past (err u3303))
(define-constant err-proposal-minimum-start-delay (err u3304))
(define-constant err-proposal-minimum-duration (err u3305))

(define-map core-team principal bool)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; --- Internal DAO functions

(define-public (set-core-team-sunset-height (height uint))
        (begin
                (try! (is-dao-or-extension))
```

```
40                  (asserts! (> height burn-block-height) err-sunset-height-in-past)
41                  (ok (var-set core-team-sunset-height height))
42          )
43  )
44
45  (define-public (set-core-team-member (who principal) (member bool))
46          (begin
47                  (try! (is-dao-or-extension))
48                  (ok (map-set core-team who member))
49          )
50  )
51
52  ;; --- Public functions
53
54  (define-read-only (is-core-team-member (who principal))
55          (default-to false (map-get? core-team who))
56  )
57
58  (define-public (core-propose (voting-contract <voting-trait>) (proposal <proposal-trait>) (st
59          (begin
60                  (asserts! (is-core-team-member tx-sender) err-not-core-team-member)
61                  (asserts! (or (is-eq (var-get core-team-sunset-height) u0) (< burn-block-heig
62                  (asserts! (>= start-burn-height (+ burn-block-height u2)) err-proposal-minimu
63                  (asserts! (>= (+ start-burn-height duration) (+ burn-block-height u72)) err-p
64                  (contract-call? voting-contract add-proposal proposal
65                          {
66                                  start-height-stacks: stacks-block-height,
67                                  start-burn-height: start-burn-height,
68                                  end-burn-height: (+ start-burn-height duration),
69                                  custom-majority: custom-majority,
70                                  proposer: tx-sender ;; change to original submitter
71                          }
72                  )
73          )
74  )
75
76  ;; --- Extension callback
77
78  (define-public (callback (sender principal) (memo (buff 34)))
79          (ok true)
80  )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 101 lines (82 loc) · 3.6 KB

Code Blame Raw

```
;; Title: BDE004 Core Execute
;; Author: Mike Cohen (based on Marvin Janssen)
;; Depends-On:
;; Synopsis:
;; This extension allows a small number of very trusted principals to immediately
;; execute a proposal once a super majority is reached.
;; Description:
;; An extension meant for the bootstrapping period of a DAO. It temporarily gives
;; some very trusted principals the ability to perform an "executive action";
;; meaning, they can skip the voting process to immediately executive a proposal.
;; The Core execute extension has an optional sunset period of ~1 month from deploy
;; time, set it to 0 to disable. The core executive team, parameters, and sunset period may b
;; by means of a future proposal.

(impl-trait .extension-trait.extension-trait)
(use-trait proposal-trait .proposal-trait.proposal-trait)

(define-data-var executive-team-sunset-height uint u0) ;; does not expire by default - can be

(define-constant err-unauthorised (err u3400))
(define-constant err-not-executive-team-member (err u3401))
(define-constant err-already-executed (err u3402))
(define-constant err-sunset-height-reached (err u3403))
(define-constant err-sunset-height-in-past (err u3404))

(define-map executive-team principal bool)
(define-map executive-action-signals {proposal: principal, team-member: principal} bool)
(define-map executive-action-signal-count principal uint)

(define-data-var executive-signals-required uint u1) ;; signals required for an executive act

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; --- Internal DAO functions
```

```
(define-public (set-executive-team-sunset-height (height uint))
	(begin
		(try! (is-dao-or-extension))
		(asserts! (> height burn-block-height) err-sunset-height-in-past)
		(ok (var-set executive-team-sunset-height height))
	)
)

(define-public (set-executive-team-member (who principal) (member bool))
	(begin
		(try! (is-dao-or-extension))
		(ok (map-set executive-team who member))
	)
)

(define-public (set-signals-required (new-requirement uint))
	(begin
		(try! (is-dao-or-extension))
		(ok (var-set executive-signals-required new-requirement))
	)
)

;; --- Public functions

(define-read-only (is-executive-team-member (who principal))
	(default-to false (map-get? executive-team who))
)

(define-read-only (has-signalled (proposal principal) (who principal))
	(default-to false (map-get? executive-action-signals {proposal: proposal, team-member
)

(define-read-only (get-signals-required)
	(var-get executive-signals-required)
)

(define-read-only (get-signals (proposal principal))
	(default-to u0 (map-get? executive-action-signal-count proposal))
)

(define-public (executive-action (proposal <proposal-trait>))
	(let
		(
			(proposal-principal (contract-of proposal))
			(signals (+ (get-signals proposal-principal) (if (has-signalled propo
		)
		(asserts! (is-executive-team-member tx-sender) err-not-executive-team-member)
		(asserts! (or (is-eq (var-get executive-team-sunset-height) u0) (< burn-block
		(and (>= signals (var-get executive-signals-required))
			(try! (contract-call? .bitcoin-dao execute proposal tx-sender))
		)
		(map-set executive-action-signals {proposal: proposal-principal, team-member:
		(map-set executive-action-signal-count proposal-principal signals)
		(ok signals)
	)
)
```

```
96
97      ;; --- Extension callback
98
99      (define-public (callback (sender principal) (memo (buff 34)))
100             (ok true)
101     )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 157 lines (128 loc) · 4.95 KB

Code Blame Raw

```
;; Title: EDE006 Treasury
;; Author: Mike Cohen (based upon work of Marvin Janssen)
;; Depends-On:
;; Synopsis:
;; A treasury that can manage STX, SIP009, SIP010, and SIP013 tokens.
;; Description:
;; An extension contract that is meant to hold tokens on behalf of the
;; DAO. It can hold and transfer STX, SIP009, SIP010, and SIP013 tokens.
;; They can be deposited by simply transferring them to the contract.
;; Any extension or executing proposal can trigger transfers.
;; Technically, the ExecutorDAO core can hold and transfer tokens
;; directly. The treasury extension merely adds a bit of separation.

(impl-trait .extension-trait.extension-trait)

(define-constant err-unauthorised (err u3000))

;; --- Transferable traits

(define-trait sip009-transferable
	(
		(transfer (uint principal principal) (response bool uint))
	)
)

(define-trait sip010-transferable
	(
		(transfer (uint principal principal (optional (buff 34))) (response bool uint
	)
)

(define-trait sip013-transferable
	(
		(transfer (uint uint principal principal) (response bool uint))
		(transfer-memo (uint uint principal principal (buff 34)) (response bool uint)
	)
)

(define-trait sip013-transferable-many
```

```
40              (
41                      (transfer-many ((list 200 {token-id: uint, amount: uint, sender: principal, r
42                      (transfer-many-memo ((list 200 {token-id: uint, amount: uint, sender: princip
43              )
44      )
45
46      ;; --- Authorisation check
47
48      (define-public (is-dao-or-extension)
49              (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
50      )
51
52      ;; --- Internal DAO functions
53
54      ;; STX
55
56      (define-public (stx-transfer (amount uint) (recipient principal) (memo (optional (buff 34))))
57              (begin
58                      (try! (is-dao-or-extension))
59                      (match memo to-print (print to-print) 0x)
60                      (as-contract (stx-transfer? amount tx-sender recipient))
61              )
62      )
63
64      (define-public (stx-transfer-many (transfers (list 200 {amount: uint, recipient: principal, m
65              (begin
66                      (try! (is-dao-or-extension))
67                      (as-contract (fold stx-transfer-many-iter transfers (ok true)))
68              )
69      )
70
71      ;; SIP009
72
73      (define-public (sip009-transfer (token-id uint) (recipient principal) (asset <sip009-transfer
74              (begin
75                      (try! (is-dao-or-extension))
76                      (as-contract (contract-call? asset transfer token-id tx-sender recipient))
77              )
78      )
79
80      (define-public (sip009-transfer-many (data (list 200 {token-id: uint, recipient: principal})))
81              (begin
82                      (as-contract (fold sip009-transfer-many-iter data asset))
83                      (ok true)
84              )
85      )
86
87      ;; SIP010
88
89      (define-public (sip010-transfer (amount uint) (recipient principal) (memo (optional (buff 34)
90              (begin
91                      (try! (is-dao-or-extension))
92                      (as-contract (contract-call? asset transfer amount tx-sender recipient memo))
93              )
94      )
95
```

```
(define-public (sip010-transfer-many (data (list 200 {amount: uint, recipient: principal, mem
        (begin
                (as-contract (fold sip010-transfer-many-iter data asset))
                (ok true)
        )
)

;; SIP013

(define-public (sip013-transfer (token-id uint) (amount uint) (recipient principal) (memo (op
        (begin
                (try! (is-dao-or-extension))
                (as-contract (match memo memo-buff
                        (contract-call? asset transfer-memo token-id amount tx-sender recipie
                        (contract-call? asset transfer token-id amount tx-sender recipient)
                ))
        )
)

(define-public (sip013-transfer-many (transfers (list 200 {token-id: uint, amount: uint, senc
        (begin
                (try! (is-dao-or-extension))
                (as-contract (contract-call? asset transfer-many transfers))
        )
)

(define-public (sip013-transfer-many-memo (transfers (list 200 {token-id: uint, amount: uint,
        (begin
                (try! (is-dao-or-extension))
                (as-contract (contract-call? asset transfer-many-memo transfers))
        )
)

;; --- Iterator functions

(define-private (stx-transfer-many-iter (data {amount: uint, recipient: principal, memo: (opt
        (begin
                (try! previous-result)
                (match (get memo data) to-print (print to-print) 0x)
                (stx-transfer? (get amount data) tx-sender (get recipient data))
        )
)

(define-private (sip009-transfer-many-iter (data {token-id: uint, recipient: principal}) (ass
        (begin
                (unwrap-panic (contract-call? asset transfer (get token-id data) tx-sender (g
                asset
        )
)

(define-private (sip010-transfer-many-iter (data {amount: uint, recipient: principal, memo: (
        (begin
                (unwrap-panic (contract-call? asset transfer (get amount data) tx-sender (get
                asset
        )
)
```

```
152
153     ;; --- Extension callback
154
155     (define-public (callback (sender principal) (memo (buff 34)))
156             (ok true)
157     )
```

radicleart first commit 133764b · 4 hours ago

Executable File · 319 lines (279 loc) · 12.2 KB

Code Blame Raw

```
;; Title: BDE021 Opinion Polling
;; Author: Mike Cohen
;; Depends-On:
;; Synopsis:
;; Enables quick opinion polling functionality.
;; Description:
;; A more streamlined type of voting designed to quickly gauge community opinion.
;; Unlike DAO proposals, opinion polls cannot change the configuration of the DAO.

(impl-trait .extension-trait.extension-trait)
(use-trait nft-trait .sip009-nft-trait.nft-trait)
(use-trait ft-trait .sip010-ft-trait.sip010-ft-trait)

(define-constant err-unauthorised (err u2100))
(define-constant err-poll-already-exists (err u2102))
(define-constant err-unknown-proposal (err u2103))
(define-constant err-proposal-inactive (err u2105))
(define-constant err-already-voted (err u2106))
(define-constant err-proposal-start-no-reached (err u2109))
(define-constant err-expecting-root (err u2110))
(define-constant err-invalid-signature (err u2111))
(define-constant err-proposal-already-concluded (err u2112))
(define-constant err-end-burn-height-not-reached (err u2113))

(define-constant structured-data-prefix 0x534950303138)
(define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
        {
                name: "BigMarket",
                version: "1.0.0",
                chain-id: chain-id
        }
    ) err-unauthorised)
))

(define-constant structured-data-header (concat structured-data-prefix message-domain-hash))
(define-constant custom-majority-upper u10000)

(define-data-var next-poll-id uint u1)
```

```
(define-map opinion-polls
        uint
        {
                market-data-hash: (buff 32),
                market-id: uint,
                votes-for: uint,
                votes-against: uint,
                start-burn-height: uint,
                end-burn-height: uint,
                proposer: principal,
    is-gated: bool,
                concluded: bool,
                passed: bool,
                custom-majority: (optional uint), ;; u10000 = 100%
        }
)
(define-map member-voted {poll-id: uint, voter: principal} bool)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

(define-public (is-core-team-member)
        (ok (asserts! (contract-call? .bitcoin-dao is-extension contract-caller) err-unauthor
)

;; --- Internal DAO functions

;; Proposals

(define-public (add-opinion-poll
    (market-data-hash (buff 32))               ;; market metadata hash
    (data {market-id: uint, start-burn-height: uint, end-burn-height: uint, custom-majority:
    (merkle-root (optional (buff 32)))      ;; Optional Merkle root for gating
  )
  (let
    (
      (poll-id (var-get next-poll-id))
    )
                (try! (is-core-team-member))

    ;; Ensure the poll does not already exist
    (asserts! (is-none (map-get? opinion-polls poll-id)) err-poll-already-exists)

    ;; Store the Merkle root if provided (gating enabled)
    (if (is-some merkle-root)
        (try! (contract-call? .bde022-market-gating set-merkle-root market-data-hash (unwrap!
        true)

    ;; Register the poll
    (map-set opinion-polls poll-id
      {market-data-hash: market-data-hash,
      market-id: (get market-id data),
      votes-for: u0,
```

```
          votes-against: u0,
          start-burn-height: (get start-burn-height data),
          end-burn-height: (get end-burn-height data),
          custom-majority: (get custom-majority data),
          proposer: tx-sender,
          concluded: false,
          passed: false,
          is-gated: (is-some merkle-root)})

        ;; Emit an event for the new poll
        (print {event: "add-poll", poll-id: poll-id, market-id: (get market-id data), market-data
        (var-set next-poll-id (+ poll-id u1))
        (ok true)
      )
    )
```

```
246           )
247         (begin
248           ;; Verify access control if the poll is gated
249           (try! (verify-access market-data-hash is-gated nft-contract ft-contract token-id proof)
250
251           ;; Ensure the voter has not already voted
252           (asserts! (is-none (map-get? member-voted {poll-id: poll-id, voter: voter})) err-alreac
253
254           ;; Ensure the voting period is active
255           (asserts! (>= burn-block-height (get start-burn-height poll-data)) err-proposal-start-r
256           (asserts! (< burn-block-height (get end-burn-height poll-data)) err-proposal-inactive)
257
258           ;; Record the vote
259           (map-set opinion-polls poll-id
260               (if for
261                   (merge poll-data {votes-for: (+ (get votes-for poll-data) u1)})
262                   (merge poll-data {votes-against: (+ (get votes-against poll-data) u1)})))
263
```

```
        ;; Mark the voter as having voted
        (map-set member-voted {poll-id: poll-id, voter: voter} true)

        ;; Emit an event for the vote
        (print {event: "poll-vote", poll-id: poll-id, voter: voter, for: for, sip18: sip18})

        (ok true)
      )
    ))

(define-read-only (verify-signature (hash (buff 32)) (signature (buff 65)) (signer principal)
        (is-eq (principal-of? (unwrap! (secp256k1-recover? hash signature) false)) (ok signer
)

(define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
        (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
)

;; Conclusion

(define-read-only (get-poll-status (poll-id uint))
    (let
        (
            (poll-data (unwrap! (map-get? opinion-polls poll-id) err-unknown-proposal))
            (is-active (< burn-block-height (get end-burn-height poll-data)))
            (passed (> (get votes-for poll-data) (get votes-against poll-data)))
        )
        (ok {active: is-active, passed: passed})
    )
)

(define-public (conclude (poll-id uint) (market-id uint))
        (let
                (
      (poll-data (unwrap! (map-get? opinion-polls poll-id) err-unknown-proposal))
      (is-active (< burn-block-height (get end-burn-height poll-data)))
                        (passed
                                (match (get custom-majority poll-data)
                                        majority (> (* (get votes-for poll-data) custom-major
                                        (> (get votes-for poll-data) (get votes-against poll-
                                )
                        )
                )
                (asserts! (not (get concluded poll-data)) err-proposal-already-concluded)
                (asserts! (>= burn-block-height (get end-burn-height poll-data)) err-end-burr
                (map-set opinion-polls poll-id (merge poll-data {concluded: true, passed: pas
                (print {event: "conclude", poll-id: poll-id, market-id: market-id, passed: pa
                (and passed (try! (contract-call? .bde023-market-staked-predictions resolve-m
                (ok passed)
        )
)

;; --- Extension callback
(define-public (callback (sender principal) (memo (buff 34)))
        (ok true)
)
```

radicleart first commit 133764b · 4 hours ago

Executable File · 145 lines (123 loc) · 5.14 KB

Code Blame Raw

```
;; Title: BDE021 Poll Gating
;; Author: Mike Cohen
;; Depends-On:
;; Synopsis:
;; Efficient verification of access control using merkel roots.
;; Description:
;; If the owner of a poll uploads a merkel root on poll creation the
;; voting contract can call into this contract to determine if the current
;; voter is allowed to vote - the rule are 1) the user must own eithr the
;; nft token or the amount of ft provided and the nft/ft contract id hash
;; must be a hash leading to the merkel root and proven by the passed in proof.

;; Define the SIP-009 and SIP-010 traits
(use-trait nft-trait .sip009-nft-trait.nft-trait)
(use-trait ft-trait .sip010-ft-trait.sip010-ft-trait)
(impl-trait .extension-trait.extension-trait)

(define-constant err-unauthorised (err u2200))
(define-constant err-either-sip9-or-sip10-required (err u2201))
(define-constant err-token-contract-invalid (err u2202))
(define-constant err-token-ownership-invalid (err u2203))
(define-constant err-expecting-nft-contract (err u2204))
(define-constant err-expecting-ft-contract (err u2205))
(define-constant err-expecting-token-id (err u2206))
(define-constant err-not-nft-owner (err u2207))
(define-constant err-not-ft-owner (err u2208))
(define-constant err-expecting-nft-buffer (err u2209))
(define-constant err-expecting-ft-buffer (err u2210))
(define-constant err-expecting-valid-merkel-proof (err u2211))
(define-constant err-expecting-merkel-root-for-poll (err u2212))
(define-constant err-expecting-an-owner (err u2213))

;; Storage: Merkle roots for each poll
(define-map merkle-roots
  (buff 32) ;; poll identifier
  (buff 32))  ;; merkel root


(define-public (is-dao-or-extension)
```

```
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
    )


    ;; Admin sets the Merkle root for a poll
    (define-public (set-merkle-root (poll-id (buff 32)) (root (buff 32)))
      (begin
        ;; Ensure only dao can set the root
        (try! (is-dao-or-extension))

        ;; Store the Merkle root
        (map-set merkle-roots poll-id root)
        (ok true)
      )
    )

    ;; Verify a Merkle proof
    (define-private (calculate-hash (hash1 (buff 32)) (hash2 (buff 32)))
      (if (< hash1 hash2)
          (sha256 (concat hash1 hash2))
          (sha256 (concat hash2 hash1))))

    (define-private (verify-merkle-proof
        (leaf (buff 32))               ;; The leaf hash (token hash)
        (proof (list 10 (buff 32)))    ;; The Merkle proof
        (root (buff 32))               ;; The Merkle root
      )
      (let
          (
            (calculated-root
              (fold calculate-hash proof leaf)
            )
          )
        (ok (is-eq calculated-root root))
      ))


    (define-private (verify-nft-ownership
        (nft-contract <nft-trait>) ;; NFT contract
        (voter principal)          ;; Voter's principal
        (token-id uint)            ;; Token ID
      )
      (let
          (
            (owner (unwrap! (contract-call? nft-contract get-owner token-id) (err u301)))
          )
        (ok (is-eq (unwrap! owner err-expecting-an-owner) voter))
      ))

    (define-private (verify-ft-balance
        (ft-contract <ft-trait>) ;; FT contract
        (voter principal)        ;; Voter's principal
        (quantity uint)          ;; Required token quantity
      )
      (let
          (
```

```
            (balance (unwrap! (contract-call? ft-contract get-balance voter) (err u304)))
          )
        (ok (>= balance quantity))
      ))


    ;; Validate proof of access
    (define-public (can-access
        (metadata-hash (buff 32))                ;; The poll ID
        (nft-contract (optional <nft-trait>)) ;; Optional NFT contract
        (ft-contract (optional <ft-trait>))   ;; Optional FT contract
        (token-id (optional uint))         ;; Token ID for NFTs
        (proof (list 10 (buff 32)))        ;; The Merkle proof
        (quantity uint)                    ;; Required token quantity
      )
      (let
          (
            ;; Determine if this is an NFT or FT contract
            (is-nft-contract (is-some nft-contract))

            ;; Fetch the Merkle root for the poll
            (root (unwrap! (map-get? merkle-roots metadata-hash) err-expecting-merkel-root-for-pc

            ;; Compute the Merkle proof leaf
            (contract-id (if is-nft-contract
                             (unwrap! (to-consensus-buff? (as-contract (unwrap! nft-contract err-
                             (unwrap! (to-consensus-buff? (as-contract (unwrap! ft-contract err-e
            (leaf (sha256 contract-id))

            ;; Verify the Merkle proof
            (proof-valid (unwrap! (verify-merkle-proof leaf proof root) err-expecting-valid-merke

            ;; Verify ownership or balance
            (ownership-valid
              (if is-nft-contract
                  (unwrap! (verify-nft-ownership (unwrap! nft-contract err-expecting-nft-contract
                  (unwrap! (verify-ft-balance (unwrap! ft-contract err-expecting-ft-contract) tx-
          )
        ;; Ensure both conditions are satisfied
        (asserts! proof-valid err-token-contract-invalid)
        (asserts! ownership-valid err-token-ownership-invalid)
        (ok true)
      ))


      ;; --- Extension callback
    (define-public (callback (sender principal) (memo (buff 34)))
            (ok true)
    )
```

radicleart first commit 133764b · 4 hours ago

312 lines (276 loc) · 10.7 KB

Code Blame Raw

```
;;;; ------------------------------------------------
;;;; my-prediction-market.clar
;;;; ------------------------------------------------
;;;; A single Clarity contract that manages multiple Yes/No
;;;; prediction markets with two fee types: 2% Dev + 2% DAO.
;;;;
;;;; Supports:
;;;; 1) Simple Stake-Based Markets
;;;; 2) (Optional) Placeholder for Shares-Based Markets
;;;; ------------------------------------------------

;; ---------------- CONSTANTS & TYPES ----------------

(define-constant DEV-FEE-BIPS u200)          ;; 2% (200 basis points)
(define-constant DAO-FEE-BIPS u200)          ;; 2% (200 basis points)

;; For demonstration, set addresses for Dev & DAO recipients
;; In production, replace with real mainnet principal addresses.
(define-constant DEV-RECIPIENT 'ST3NBRSFKX28FQ2ZJ1MAKX58HKHSDGNV5N7R21XCP)
(define-constant DAO-RECIPIENT 'STNHKEPYEPJ8ET55ZZ0M5A34J0R3N5FM2CMMMAZ6)
;;(define-constant DEV-RECIPIENT 'ST3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNZN9J752)
;;(define-constant DAO-RECIPIENT 'ST167Z6WFHMV0FZKFCRNWZ33WTB0DFBCW9M1FW3AY)

;; Market Types (0 => Stake-based, 1 => Shares-based)
(define-constant MARKET_TYPE_STAKE u0)
(define-constant MARKET_TYPE_SHARES u1)

(define-constant err-unauthorised (err u10000))
(define-constant err-invalid-market-type (err u10001))
(define-constant err-amount-too-low (err u10002))
(define-constant err-wrong-market-type (err u10003))
(define-constant err-already-concluded (err u10004))
(define-constant err-market-not-found (err u10005))
(define-constant err-user-not-winner (err u10006))
(define-constant err-not-participant-or-invalid-market (err u10007))
(define-constant err-user-balance-unknown (err u10008))
(define-constant err-market-not-concluded (err u10009))
(define-constant err-not-implemented (err u10010))
(define-constant err-insufficient-balance (err u10011))
```

```
(define-constant err-insufficient-contract-balance (err u10012))
(define-constant err-user-share-is-zero (err u10013))
(define-constant err-dao-fee-is-zero (err u10014))

;; Data structure for each Market
;; market-id: internal numeric ID
;; creator: who created the market
;; market-type: 0 => stake-based, 1 => shares-based
;; yes-pool/no-pool: total amount staked (or pooled) on Yes/No
;; concluded: whether the market is concluded
;; outcome: true => Yes won, false => No won (only valid if concluded = true)
(define-map markets
  uint
  {
                metadata-hash: (buff 32),
    creator: principal,
    market-type: uint,
    yes-pool: uint,
    no-pool: uint,
    concluded: bool,
    outcome: bool
  }
)

;; For stake-based, we simply track how much each user staked on Yes/No.
;; For shares-based, you'd track how many shares the user holds.
(define-map stake-balances
  { market-id: uint, user: principal }
  {
    yes-amount: uint,
    no-amount: uint
  }
)
(define-data-var market-counter uint u0)

;; ---------------- PUBLIC FUNCTIONS ----------------
(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bitcoin-dao) (contract-call? .bitcoin-dao is-exte
)

;; Creates a new Yes/No prediction market. `mtype` can be 0 (stake-based)
;; or 1 (shares-based). Returns the new market-id.
(define-public (create-market (mtype uint) (metadata-hash (buff 32)))
  (begin
    (asserts! (or (is-eq mtype MARKET_TYPE_STAKE) (is-eq mtype MARKET_TYPE_SHARES)) err-inval

    (let ((new-id (var-get market-counter)))
      (map-set markets
        new-id
        {
          metadata-hash: metadata-hash,
          creator: tx-sender,
          market-type: mtype,
          yes-pool: u0,
          no-pool: u0,
          concluded: false,
```

```
96                outcome: false
97              }
98            )
99            ;; Increment the counter
100           (print {event: "create", market-id: new-id, metadata-hash: metadata-hash, market-type:
101           (var-set market-counter (+ new-id u1))
102           (ok new-id)
103         )
104       )
105     )
106
107     (define-read-only (get-market-data (market-id uint))
108             (map-get? markets market-id)
109     )
110
```

```
            (transfer-amount (- amount fee))
          )
        (begin
          ;; Ensure amount is valid
          (asserts! (>= amount u5000) err-amount-too-low)
          ;; Check tx-sender's balance
          (asserts! (>= sender-balance amount) err-insufficient-balance)
          ;; Transfer STX to the contract
          (try! (stx-transfer? transfer-amount tx-sender .bde023-market-staked-predictions))
          ;; Transfer the fee to the dev fund
          (try! (stx-transfer? fee tx-sender DEV-RECIPIENT))
          ;; Verify the contract received the correct amount
          (asserts! (>= (stx-get-balance .bde023-market-staked-predictions) transfer-amount) err-

          (ok transfer-amount)
        )
      )
    )

    (define-private (calculate-fee (amount uint) (fee-bips uint))
      (let ((fee (/ (* amount fee-bips) u10000)))
        fee
      )
    )

```

```
264
265
266     (define-private (take-fee (amount uint) (fee-bips uint))
267       (let (
268             (fee (/ (* amount fee-bips) u10000))
269            )
270         fee
271       )
272     )
273
274     (define-private (claim-winnings-internal
275       (market-id uint)
276       (user-stake uint)
277       (winning-pool uint)
278       (total-pool uint)
279       (yes-won bool))
280       (let (
281             (original-sender tx-sender)
282             (user-share (/ (* user-stake total-pool) winning-pool))
283             (dao-fee (/ (* user-share DAO-FEE-BIPS) u10000))
284             (user-share-net (- user-share dao-fee))
285         )
286         (begin
287           ;; Ensure inputs are valid
288           (asserts! (> user-share-net u0) err-user-share-is-zero)
289           (asserts! (> dao-fee u0) err-dao-fee-is-zero)
290
291           ;; Perform transfers
292           (as-contract
293             (begin
294               ;; Transfer user share, capped by initial contract balance
295               (try! (stx-transfer? user-share-net tx-sender original-sender))
296               (try! (stx-transfer? dao-fee tx-sender DAO-RECIPIENT))
297             )
298           )
299
300           ;; Zero out user stake
301           (map-set stake-balances { market-id: market-id, user: tx-sender }
302             {
303               yes-amount: u0,
304               no-amount: u0
305             })
306
307           ;; Log and return user share
308           (print {event: "claim", market-id: market-id, is-won: yes-won, claimer: tx-sender, user
309           (ok user-share-net)
310         )
311       )
312     )
```

radicleart first commit 133764b · 4 hours ago

84 lines (67 loc) · 2.86 KB

Code Blame Raw

```
;; Title: EDE005 Dev Fund
;; Author: Marvin Janssen
;; Depends-On: EDP000
;; Synopsis:
;; A simple pre-seeded dev fund that can pay out developers on a monthly basis.
;; Description:
;; Initialised by EDP001 Dev Fund. Developers can be awarded a monthly allowance
;; and can claim it from this extension. Principals can be added and removed, and
;; allowances can be changed via future proposals.

(impl-trait .extension-trait.extension-trait)

(define-constant one-month-time u4380) ;; 43,800 minutes / 10 minute average block time.

(define-constant err-unauthorised (err u3000))
(define-constant err-no-allowance (err u3001))
(define-constant err-already-claimed (err u3002))

(define-map monthly-developer-allowances principal {start-height: uint, allowance: uint})
(define-map claim-counts principal uint)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .executor-dao) (contract-call? .executor-dao is-ex
)

;; --- Internal DAO functions

(define-public (set-developer-allowance (start-height uint) (allowance uint) (who principal))
        (begin
                (try! (is-dao-or-extension))
                (ok (map-set monthly-developer-allowances who {start-height: start-height, al
        )
)

(define-private (set-developer-allowances-iter (item {start-height: uint, allowance: uint, wh
        (map-set monthly-developer-allowances (get who item) {start-height: (get start-height
)
```

```
40
41  (define-public (set-developer-allowances (developers (list 200 {start-height: uint, allowance
42          (begin
43                  (try! (is-dao-or-extension))
44                  (ok (fold set-developer-allowances-iter developers true))
45          )
46  )
47
48  (define-public (transfer (amount uint) (recipient principal) (memo (optional (buff 34))))
49          (begin
50                  (try! (is-dao-or-extension))
51                  (as-contract (contract-call? .ede000-governance-token transfer amount tx-send
52          )
53  )
54
55  ;; --- Public functions
56
57  (define-read-only (get-developer-allowance (who principal))
58          (map-get? monthly-developer-allowances who)
59  )
60
61  (define-read-only (get-developer-claim-count (who principal))
62          (default-to u0 (map-get? claim-counts who))
63  )
64
65  (define-public (claim (memo (optional (buff 34))))
66          (let
67                  (
68                          (entry (unwrap! (get-developer-allowance tx-sender) err-no-allowance)
69                          (claim-count (get-developer-claim-count tx-sender))
70                          (start-height (get start-height entry))
71                          (max-claims (/ (- block-height start-height) one-month-time))
72                          (developer tx-sender)
73                  )
74                  (asserts! (< claim-count max-claims) err-already-claimed)
75                  (map-set claim-counts tx-sender max-claims)
76                  (as-contract (contract-call? .ede000-governance-token transfer (* (- max-clai
77          )
78  )
79
80  ;; --- Extension callback
81
82  (define-public (callback (sender principal) (memo (buff 34)))
83          (ok true)
84  )
```