ErgoNames

A Decentralized Naming Protocol on Ergo

Version 1.0 — Draft

June 2026

ergonames.io

Contracts: github.com/ergonames/ergonames-contracts

Abstract

ErgoNames is a decentralized naming protocol built natively on the Ergo blockchain. It maps human-readable names (e.g. `~adoo`) to Ergo addresses and arbitrary on-chain identities, with each name issued as a non-fungible token (NFT) under a *lifetime-ownership* model: a name is bought once, owned forever, and freely transferable — there are no renewal fees and no expiry.

The protocol is implemented entirely in ErgoScript on Ergo's extended-UTXO (eUTXO) model. A single on-chain registry box, carrying an authenticated AVL tree, guarantees global name uniqueness without any trusted intermediary. Registration uses a commit–reveal scheme to prevent front-running, prices names in USD via a decentralized ERG/USD oracle, and embeds the name's artwork directly on-chain as an SVG — no external hosting, IPFS, or mutable metadata. Every name additionally anchors its own subname registry, enabling hierarchical names (`sub~name`) controlled by the parent name's holder.

This paper describes the protocol's design goals, contract architecture, registration flow, pricing mechanism, refund and fund-safety guarantees, and upgrade path.

Contents

1 Introduction

1.1 Motivation

Blockchain addresses are long, unmemorable, and error-prone. A naming layer — analogous to DNS for the internet — is foundational infrastructure for payments, identity, and applications. Ethereum's ENS demonstrated demand for such a layer, but its design carries assumptions that do not fit every chain or every user:

- **Rental, not ownership.** ENS names expire and must be renewed; a lapsed renewal can mean losing an identity (and everything pointed at it) to a squatter.
- **Account-model implementation.** ENS relies on mutable contract storage and a resolver-contract hierarchy that has no direct analogue in UTXO systems.

ErgoNames takes the opposite stance on the first point and solves the second natively: names are *owned for life*, and the entire registry lives in Ergo's eUTXO model using on-chain authenticated data structures rather than contract storage.

1.2 Design goals

1. **Lifetime ownership.** One purchase, permanent ownership. A name is an NFT in the user's wallet; holding the token *is* owning the name.
2. **Global uniqueness, trustlessly enforced.** No two identical names can ever be minted. Uniqueness is enforced by consensus, not by an off-chain operator.
3. **Front-running resistance.** Observing a pending registration in the mempool must not allow an attacker to steal the name.
4. **Fund safety at every step.** Registration is a multi-transaction process; at every intermediate stage the user's funds must be recoverable by the user alone, without operator cooperation.
5. **Fully on-chain artifacts.** Name metadata and artwork are stored on-chain. Nothing about a name depends on a server staying up.
6. **Stable, predictable pricing.** Names are priced in USD by length, converted to ERG at mint time via a decentralized oracle.
7. **Upgradability without custody.** Protocol parameters and contract logic can evolve via a multisig escape hatch — but the operator can never confiscate, modify, or expire an already-minted name, because minted names are ordinary tokens in user wallets.

2 Background: Ergo and the eUTXO Model

Ergo is a proof-of-work blockchain whose transaction model extends Bitcoin's UTXO design. Key primitives used by ErgoNames:

- **Boxes.** The unit of state. A box holds ERG, tokens, and up to ten typed registers (R4–R9), and is guarded by an ErgoScript contract that must evaluate to true for the box to be spent.
- **Tokens and singletons.** Ergo supports native tokens minted in-transaction. A token with supply 1 (a *singleton*) uniquely identifies a logical entity across box generations: when the registry box is spent and recreated, the singleton proves the new box is the same registry.

- **Authenticated AVL trees.** Ergo natively supports cryptographically authenticated dictionaries (AVL+ trees). A box register stores only the tree's digest (≈33 bytes); off-chain actors supply insertion/lookup proofs that the contract verifies on-chain. This gives ErgoNames an unbounded key–value registry with constant on-chain footprint.
- **Data inputs.** Transactions can *reference* boxes without spending them. ErgoNames reads the ERG/USD oracle this way, so price feeds are consumed without contention.
- **Sigma propositions.** Spending conditions are sigma-protocol statements, enabling native threshold signatures (`atLeast(k, ...)`) used for the protocol's multisig escape hatches.

These primitives are the reason ErgoNames needs no resolver-contract hierarchy: the registry is one box, the proof system enforces uniqueness, and ownership is plain token custody.

3 Protocol Overview

3.1 Names

An ErgoName is a string of one or more characters drawn from `[a-z A-Z 0-9 _]` (the X/Twitter handle alphabet), rendered with a leading tilde by convention: `~adoo`. Validity is enforced **on-chain** — the registry contract checks every byte of the name against the allowed ASCII ranges before permitting a mint.

A registered name is an EIP-4[1] picture NFT belonging to the official ErgoNames collection (EIP-34). Its artwork — a rendered card showing the name — is generated deterministically and embedded in the token's registers as a base64 `data:image/svg+xml` URI together with its SHA-256 hash. The NFT is self-contained: wallets and marketplaces can display it with zero external dependencies, forever.

3.2 Resolution and reverse resolution

- **Forward resolution** (`~name` → address): the current holder of the name token *is* the resolved address. Resolution is therefore live and automatic — transferring the NFT re-points the name, with no separate "set resolver" step and no stale records.
- **Reverse resolution** (address → names): an indexer enumerates names held by an address, with a user-designated primary name.

Because resolution is defined by token custody, it inherits all of Ergo's wallet semantics: names can sit in cold storage, multisig vaults, or smart contracts, and "who owns this name" is always answerable from the UTXO set alone.

3.3 On-chain components

The protocol consists of a small set of long-lived singleton boxes and per-registration ephemeral boxes (Table 1).

The registry's AVL tree maps blake2b256(*name*) → *name token ID*. Inserting a key that already exists is cryptographically impossible to prove, so **uniqueness is a consequence of the proof system itself**, not of any operator behaving honestly.

[1] `https://github.com/ergoplatform/eips/blob/master/eip-0004.md`

Component	Kind	Role
Registry	singleton box	Authoritative name set (AVL tree digest), price map, commit-age window; validates every mint
Collection	singleton box	Holds the EIP-34 collection token supply; exactly one collection token is consumed per minted name, tying every name verifiably to the official collection
Config	singleton box	On-chain updatable parameter store (AVL tree), reserved for alternative payment options
Subname registry	one singleton per name	Anchors the subname tree for each registered name (§6)
Fee split	accumulator contract	Receives mint fees; enforces per-mille distribution to configured recipients
Commit / Reveal-proxy / Reveal	ephemeral boxes	Per-registration boxes implementing the commit–reveal flow (§4)

Table 1: On-chain components of the ErgoNames protocol.

4 Registration: The Commit–Reveal Mint Flow

Naïve on-chain registration is front-runnable: an attacker watching the mempool sees the desired name and submits a competing registration with a higher fee. ErgoNames eliminates this with a two-phase commit–reveal scheme split across four transactions. The first two are built and signed **by the user in their own wallet**; the last two are executed by a permissionless off-chain transaction operator ("bot").

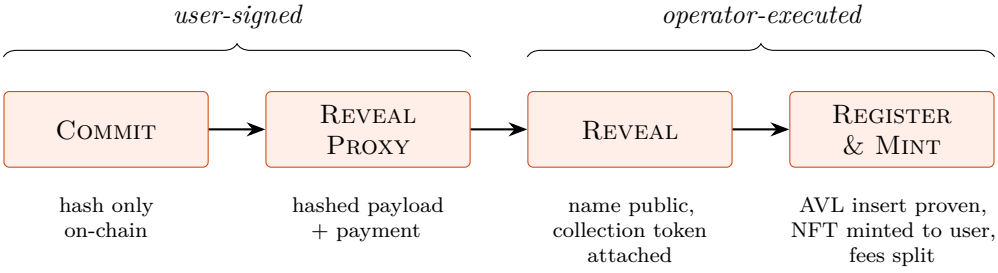

Figure 1: The four-transaction registration flow. The user signs the first two transactions in their own wallet; the off-chain operator executes the last two against the registry.

1. Commit. The user creates a commit box containing their public key and the commitment

$$\text{blake2b256}(\mathit{secret} \,\|\, \mathit{userPK} \,\|\, \mathit{nameBytes}).$$

Nothing about the name is revealed. The registry enforces a commit **age window** (minAge, maxAge): the commit must mature for a minimum number of blocks before it can be consumed (so an attacker cannot observe a reveal and out-commit it retroactively), and expires after a maximum age.

2. Reveal proxy. The user funds a proxy box carrying the full registration payment plus operator/miner fees, and a binding hash of the exact reveal box to be created. The plaintext name still does not appear on-chain; it travels to the operator off-chain.

3. Reveal. The operator spends the proxy together with the collection box, producing a reveal box that now carries the name, the user's secret, the commit box reference, the EIP-4 artwork registers, and **one collection token** borrowed from the collection box. The reveal box's contents are constrained byte-exactly by the hash committed in step 2 — the operator cannot alter the name, the recipient, or the fees.

4. Register & mint. The final transaction spends the reveal, registry, and commit boxes together. The registry contract verifies, in consensus:

- the commit hash matches blake2b256($secret \,\|\, userPK \,\|\, nameBytes$) — proving this user committed to this name before revealing it;
- the commit box age lies within the registry's age window;
- the name is valid ASCII (§3.1);
- the supplied **AVL insertion proof** transitions the registry digest by inserting exactly blake2b256($name$) $\rightarrow$ $tokenId$, which fails if the name already exists;
- the payment covers the oracle-derived price (§5) and is routed to the fee-split contract;
- the new name NFT (carrying the on-chain SVG and collection linkage) is delivered to the committed user's public key;
- a fresh **subname registry** box is created for the name (§6);
- the recreated registry box preserves the singleton, the updated tree, and all parameters.

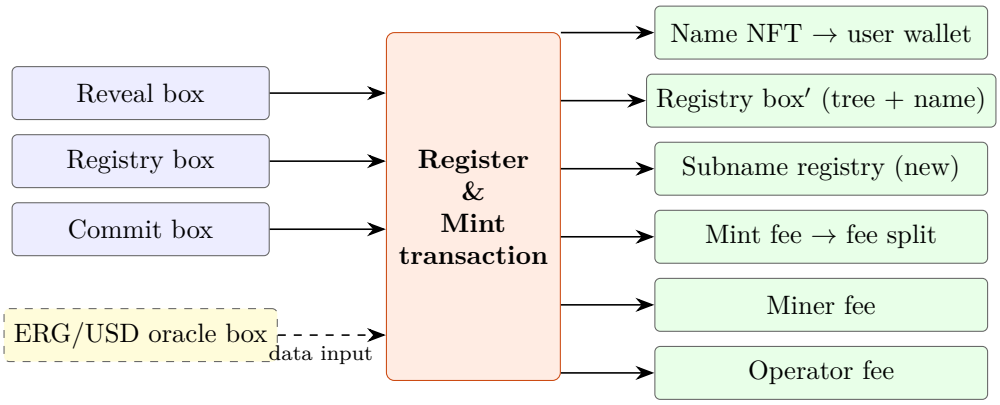

Figure 2: Anatomy of the register & mint transaction. The oracle box is referenced read-only (data input), so price reads never contend with other transactions. The registry box is recreated with the singleton, updated AVL tree, and unchanged parameters.

The token is minted with two units: one to the user's wallet (the ownership token) and one bound into the name's subname registry box, permanently linking the name to its subname tree.

Front-running analysis. Before step 4, the name appears on-chain only as hashes. An attacker who learns the name at reveal time cannot register it, because they cannot produce a commit box older than *minAge* blocks that hashes to a commitment over *their* key — the age window makes retroactive commitment impossible. Two honest users racing for the same name are serialized by the registry singleton: exactly one insertion proof can succeed, and the loser is refunded (§7).

5 Pricing

Names are priced **in USD by character length**, with shorter names costing more, and settled in ERG at the prevailing market rate:

- The registry box carries a **price map**: a vector where index n is the USD price for an n-character name (the last entry covers all longer names).
- At mint time, the transaction must reference the **SigUSD ERG/USD oracle pool box as a data input**. The registry computes $fee = nanoErgPerUsd \times price$ from the oracle's current datapoint and requires the payment output to cover it (within a bounded slippage window, since the oracle may update between quote and confirmation).
- The price map is an on-chain register, updatable by the protocol multisig via the escape hatch (§8) **without redeploying the registry** — pricing policy can evolve while the name set, singleton, and history remain untouched.

Because the oracle is a data input, price reads are contention-free, and because the conversion happens inside the contract, the operator cannot overcharge: a user who pays the on-chain-computed fee gets the name, period. The architecture also reserves support (via the config box and oracle data inputs) for payment in tokens other than ERG.

There are **no recurring fees**. The mint fee is the only payment a name holder ever makes.

6 Subnames

Every registered name owns a hierarchical namespace beneath it. The mint transaction creates a dedicated **subname registry** box for the new name — a singleton carrying its own AVL tree and a unit of the parent name's token, cryptographically binding the subname tree to the parent.

The holder of the parent name NFT can:

- **mint subnames** (`pay`, `vault`, ... under `~adoo`), each itself an NFT with its own child subname registry — the hierarchy nests to arbitrary depth;
- **revoke a child subname** (parent burns child), preserving the parent's authority over its namespace;

and a subname holder can **self-burn** their subname. All three operations are enforced by the subname registry contract with the same AVL-proof machinery as the root registry: uniqueness within each namespace level is consensus-enforced.

This supports organizational identity (`alice` under `~mycompany`), service endpoints, and delegated naming, while keeping the root registry's economics intact.

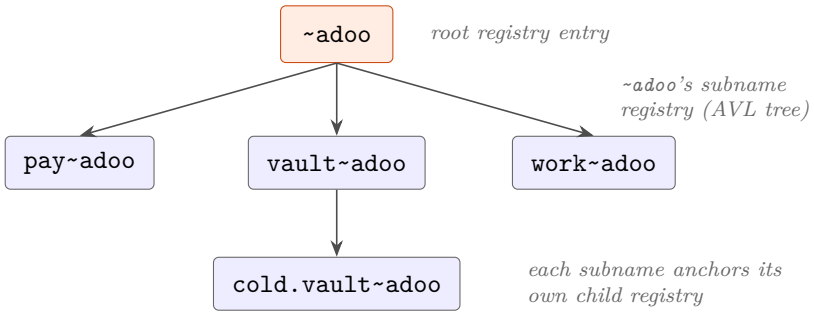

Figure 3: Hierarchical subnames. Each registered name carries a singleton subname registry; every subname is itself an NFT that anchors a child registry, nesting to arbitrary depth. The parent's holder can mint and revoke children.

7 Fund Safety and Refunds

Registration spans multiple transactions, so the protocol is explicit about every intermediate state. **Each ephemeral box is refundable by the user's signature alone** — the operator can never strand or seize in-flight funds:

- **Commit refund.** After the commit's maximum age passes, the user can reclaim the commit box to their own key.
- **Reveal-proxy refund.** The user can unwind an unprocessed proxy box, recovering the full payment.
- **Reveal refund.** If a reveal can no longer complete (e.g. the name was registered first by someone else), the user reclaims the reveal box's value; the transaction simultaneously **returns the borrowed collection token** to the collection box, preserving the collection-supply invariant.

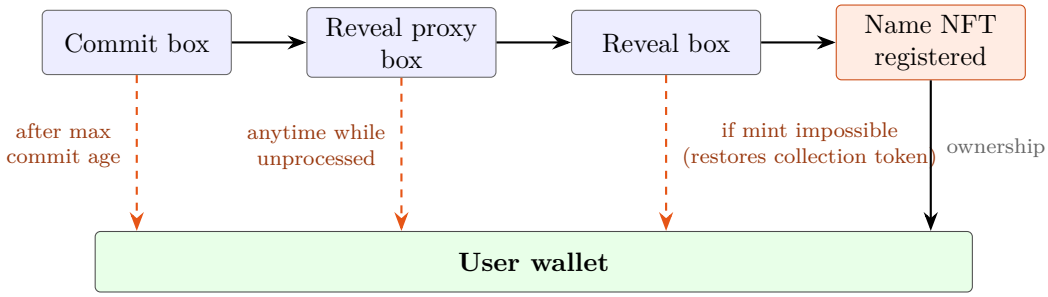

Figure 4: Refund paths (dashed). Every intermediate box is recoverable by the user's signature alone; the operator can neither strand nor seize in-flight funds. The successful path ends with the name NFT in the same wallet.

Conversely, refunds *require* the user's signature: the operator cannot redirect a refund to itself. Races for the same name therefore resolve cleanly — one party gets the name, the other gets their money back, and no third outcome exists.

The collection token supply decreases by exactly one per successful mint and is restored by every refund, making the total number of names ever minted auditable directly from the collection box.

8 Governance and Upgradability

ErgoNames separates what must be immutable from what must be able to evolve:

- **Minted names are beyond protocol control.** A name NFT is an ordinary token in a user wallet. No multisig, upgrade, or migration can confiscate, alter, or expire it. Lifetime ownership is structural, not a policy promise.
- **Protocol boxes carry a multisig escape hatch.** The registry, subname registries, collection, config, and fee-split boxes can each be spent by a threshold signature of the protocol keys — but **only in transaction shapes that are not mints**. The contracts structurally distinguish mint-shaped transactions (which must satisfy the full validation logic regardless of any signature) from migration-shaped ones (which fall through to the multisig). The multisig therefore cannot be used to mint names that bypass validation; it can only re-point a box to revised contract logic.
- **Upgrades preserve state.** Because the registry's identity is its singleton token — not its script hash — a migration moves the AVL tree, price map, and history into a box with new logic while every existing name, proof, and reference remains valid. Parameter changes (pricing, fee recipients, age window) are register-level updates requiring no logic change at all.
- **Fee distribution is contract-enforced.** Mint fees accumulate in the fee-split contract, spendable only by a transaction paying every configured recipient at least its per-mille share. Revenue policy (e.g. a future dev/treasury/DAO split) is a config change, not a custody change.

This yields a credible path from the current founder-operated multisig toward progressively decentralized key holding, without ever touching user-owned names.

9 Off-Chain Architecture

The chain enforces all rules; off-chain components only provide convenience and liveness. The reference stack:

- **Transaction operator (bot).** A stateless-by-design daemon that watches pending registrations and executes the reveal and register transactions. It maintains an ordered insert journal mirroring the registry AVL tree (fully reconstructible by replaying the chain) to generate insertion proofs. The operator is **permissionless in effect**: it can neither alter a registration (reveal contents are hash-bound), steal funds (refunds are user-signed), nor censor durably (anyone with the public contracts can run a competing operator against the same registry).
- **Indexer & resolution API.** Follows the registry singleton's box chain, detecting mints structurally, and serves forward/reverse resolution, ownership, and pricing queries. Any party can run one; the chain is the source of truth.
- **Web frontend.** A wallet-connected dApp (Nautilus) that performs name search, builds and signs the user-side transactions (commit, reveal-proxy) entirely in the browser, tracks registration status, and exposes self-service refund tooling for any stuck state. The frontend computes no protocol-critical cryptography beyond standard transaction building; the byte-exact reveal-box hashing is served by the operator API and verified by the contracts on-chain.

A dedicated Ergo full node backs the operator, with the operator's API and the resolution API exposed publicly behind TLS and rate limiting. Figure 5 shows the reference deployment.

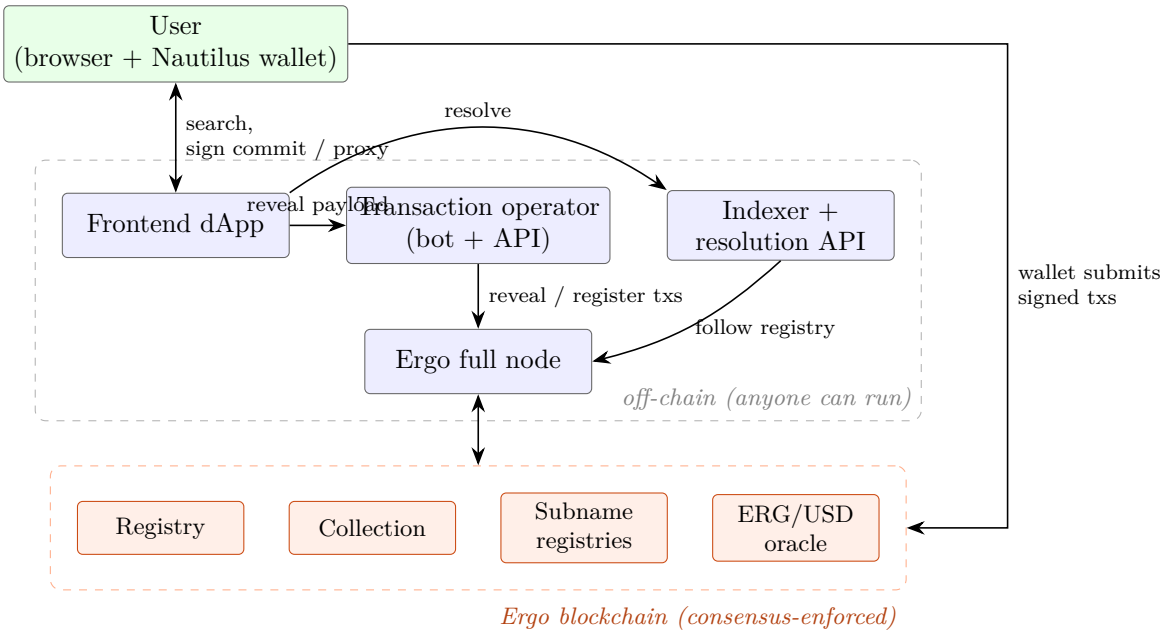

Figure 5: Reference architecture. The chain enforces all protocol rules; off-chain components provide only convenience and liveness, and any party can run their own operator or indexer against the same on-chain registry.

10 Security Considerations

- **Front-running** is mitigated by commit–reveal with an enforced minimum commit age (§4).
- **Registry contention** is inherent to a singleton design: only one mint can advance the registry per transaction. The operator serializes registrations; throughput is bounded by block capacity, which is ample for a naming workload. Losers of any race are refunded.
- **Oracle dependence.** Pricing trusts the SigUSD oracle pool, Ergo's longest-running decentralized oracle. A bounded slippage window absorbs datapoint drift between quote and confirmation; the oracle is read-only (data input), so it cannot be manipulated within the mint transaction itself.
- **Operator compromise** cannot mint invalid names (consensus-validated), steal in-flight funds (user-signed refunds), or alter registrations (hash-bound reveals). The blast radius of a compromised operator key is limited to its own fee revenue and liveness.
- **Multisig compromise** could re-point protocol boxes to malicious logic for *future* mints, but cannot touch existing names or in-flight user refund paths. Key-role separation (cold multisig distinct from the operator's hot key) is part of the launch hardening.
- **Name squatting** is an economic question rather than a protocol flaw under lifetime ownership; the length-tiered USD pricing makes mass-squatting short names expensive. The price map remains tunable as the market reveals itself.

- **Homoglyph/confusable attacks** are constrained by the deliberately minimal ASCII alphabet — no Unicode, no mixed scripts, no punycode ambiguity.

11 Comparison with Existing Naming Systems

	ErgoNames	**ENS (Ethereum)**	**DNS**
Ownership model	Lifetime, one-time fee	Rental, annual renewal	Rental, annual renewal
Expiry risk	None	Name lost if renewal lapses	Name lost if renewal lapses
Uniqueness enforcement	Consensus (AVL proofs)	Contract storage	Registrar hierarchy
Resolution	Token custody = ownership = resolution	Separate resolver records	Zone files
Artwork/metadata	Fully on-chain SVG	Typically off-chain	n/a
Subnames	Native, NFT per subname, parent-revocable	Supported via wrappers	Delegation
Censorship of registration	Operator bypassable; rules consensus-enforced	Permissionless	Registrar policy

Table 2: ErgoNames compared with ENS and DNS.

12 Roadmap

1. **Launch (mainnet).** Fresh production genesis; public minting via the web dApp; resolution and reverse-resolution APIs; self-service refund tooling. The full mint, refund, pricing, fee-distribution, and upgrade machinery described in this paper is implemented and has been exercised end-to-end on Ergo mainnet.
2. **Ecosystem integration.** Wallet-native resolution (send-to-`~name`), reverse resolution as display identity, marketplace listing of the collection.
3. **Subname tooling.** UI for minting, delegating, and revoking subnames.
4. **Payment options.** Token-denominated payment via the config box and DEX price data inputs.
5. **Progressive decentralization.** Expanded multisig key set and role separation; published runbooks enabling third-party operators and indexers.

13 Conclusion

ErgoNames demonstrates that a complete naming protocol — unique registration, front-running resistance, oracle pricing, hierarchical subnames, contract-enforced revenue splits, and safe upgradability — can be expressed in the eUTXO model with a footprint of one registry box and a handful of small contracts. The design inverts the prevailing rental model: a name is property, not a subscription. Ownership is token custody, resolution is a consequence of ownership, and every artifact a name consists of lives on-chain. The result is a naming layer with the permanence users expect from the word *name*.

Contracts: github.com/ergonames/ergonames-contracts (ErgoScript v1, source-available). Original contract author: Luca D'Angelo. This document describes protocol version 1.