Compliant Settlement Layer and Tokenisation Engine

An on-chain approved-holder registry for DataVault's real-world asset exchange, delivered as an API service behind DataVault's existing API layer.

Prepared for: DataVault AI Inc. (NASDAQ: DVLT)

Prepared by: Mark Abbott, Co-Founder, RubySDK

Date: July 2026

Classification: Confidential

> This proposal replaces the RubySDK privacy proposal of February 2026, the updated proposal of 18 May 2026, and the Rubytech counterproposal of March 2026. Where those documents remain relevant, the material is carried forward here. Where they have been overtaken by the scope set out here, this document governs.

1 The engagement

This proposal covers a compliant settlement layer for DataVault's real-world asset exchange: an on-chain registry of approved holders, permissioned token contracts that enforce it, and an API through which the DataVault platform deploys and manages both.

The asset class

The engine is built for pre-extraction mining and energy assets: gold, copper, diamond, rare earths, and geothermal. These are tokenised before the resource comes out of the ground, so the token represents an entitlement against an asset that will be produced rather than one already held in a vault.

The registry itself is agnostic to the underlying asset. It governs who may hold a token and under what conditions, so any asset DataVault chooses to issue as a security uses the same infrastructure. The assets above are the first application, not the boundary of it.

What the asset class requires

This proposal is built for instruments treated as securities rather than commodities. RubySDK expresses no view on that classification and does not advise on it; we supply the infrastructure the classification requires.

What follows from it is structural. Where an asset is treated as a security, the register of holders is maintained on chain, where the asset settles, rather than on the platform that issued it. Once a token can move from one wallet to another without passing back through a platform's front end, an off-chain register no longer determines ownership.

In practice that means one specific artefact. An on-chain list of approved and restricted holders, enforced at the contract, so that a transfer to an unapproved wallet fails whether or not the trade was initiated on DataVault's platform. Everything about it is standardised, and section 2 sets out the standard.

The boundary

The boundary this proposal assumes is that DataVault retains the exchange, the client relationships, the assets, the tokenomics, the metadata storage and analytics, and the chain publishing. RubySDK supplies the registry, the permissioned token contracts, the redemption logic, and the API that connects them. Nothing in this proposal asks DataVault to replace the portal, the API layer, or the trading platform.

2 What RubySDK provides

RubySDK issues permissioned security tokens using ERC-3643, the T-REX standard, and this proposal is built to it. It is not a RubySDK invention and it is not a proprietary variant: it is the token standard the regulated tokenisation market has converged on, and it exists precisely to solve the problem of an on-chain holder list.

The four parts of the standard

Component	What it does
Identity registry	The on-chain list of who is permitted to hold the token. Each approved holder has an on-chain identity contract bound to their wallet address. This is the artefact an off-chain holder register cannot provide.
Claims and trusted issuers	An identity carries signed claims: verified, jurisdiction, accreditation status, expiry. Claims are issued by a party the token issuer trusts. This proposal assumes DataVault, or DataVault's KYC provider, acts as that trusted issuer.
Compliance modules	Transfer rules attached to the token: holder caps, jurisdiction restrictions, lock-up periods, maximum holding size, transfer freezes. Rules are configured per token, not written from scratch per asset.
Token contract	Checks the registry and the compliance modules before every transfer. If the receiving wallet is not a verified, eligible holder, the transfer reverts.

Unapproved trades

An attempted trade by an unapproved party needs no custom on-chain logic beyond rejecting it. Rejection is the standard behaviour of the compliance module. The transfer function calls a check, the check fails, and the transaction reverts with a reason code. There are no custom rules to write for invalid trades, and DataVault's platform can read the revert reason to show the user why the trade did not settle.

Where KYC sits

KYC stays with DataVault. RubySDK does not perform identity verification and does not want DataVault's customer records. What goes on chain is the verified claim, not the personal data: a signature from the trusted issuer asserting that the wallet behind a given identity passed verification, with a claim topic and an expiry. No name, no document, no date of birth, no address. That is what makes user-to-user transfer possible without moving personal data on chain.

> **Recovery and control.** The standard includes agent roles for the issuer: freezing an address or a partial balance, forcing a transfer where a court or regulator requires it, and recovering tokens to a new wallet when a holder loses their keys. These are contract functions, available to DataVault as issuer, and they are part of what makes the token defensible as a security rather than a bearer instrument.

3 The custom layer: redemption against the underlying asset

The one piece that is genuinely custom for a pre-extraction asset is redemption. A token issued before the resource is out of the ground has nothing to redeem against until the mine is in operation, so the token needs a defined period during which a holder may redeem against the underlying asset once production begins. RubySDK proposes to implement this as a redemption period on the token.

This is an extension to the compliance and transfer-restriction layer, not a rewrite of it. The redemption period is a state on the token that the compliance module reads, in the same way it reads a lock-up. The design below is RubySDK's proposal, implemented against DataVault's commercial rules and confirmed in Discovery.

Function	Behaviour
`setRedemptionPeriod`	Defines the redemption period for a token: start, end, redemption ratio, and any per-holder or aggregate cap. Issuer role only.
`openRedemptionPeriod`	Opens redemption when the mine begins operating. Emits an event DataVault's platform can subscribe to.
`closeRedemptionPeriod`	Closes the period. Redemption calls after close revert.
`getRedemptionPeriod`	Read-only. Returns current period state, remaining capacity, and open or closed status.
`redeem`	Burns the holder's tokens against the underlying asset entitlement and emits a redemption event carrying the holder identity, quantity, and reference for DataVault's settlement records.

Where DataVault already holds redemption logic of its own, we take that as the input specification rather than write our own from an assumption. If the existing implementation is sound, we adapt it to sit behind the compliance module rather than replace it. Where existing code assumes an unrestricted token, we will say so in writing during Discovery and propose the minimum change.

4 Integration architecture

The integration assumes a web portal in front of an API layer through which third-party services extend the platform, which is the shape most exchange platforms take. RubySDK becomes one of those third-party services. The integration is machine to machine over JSON and HTTPS, and it does not require DataVault to change the portal, the API layer, or the exchange. The specifics of that layer are confirmed in Discovery rather than assumed here.

The calls

Call	Purpose and return
`POST /v1/tokens`	Request deployment of an asset token. DataVault sends asset name, symbol, decimals, supply, target chain, compliance ruleset, and redemption rules. Returns a request identifier.
`GET /v1/tokens/{id}`	Returns deployment state and, once deployed, the token contract address, the identity registry address, the deployment transaction hash, the chain identifier, and the contract interface. DataVault verifies all of it directly on chain.
`POST /v1/identities`	Register an approved holder: DataVault's internal user reference, wallet address, and the KYC claim attestation from DataVault or its provider. Returns the on-chain identity address and the transaction hash.
`PATCH /v1/identities/{id}`	Update a holder's claims: jurisdiction change, accreditation renewal, expiry extension.
`DELETE /v1/identities/{id}`	Revoke a holder. The address is removed from the registry, which removes eligibility to receive the token. Revocation does not act on a balance the holder already holds. Treatment of a revoked holder's existing balance follows the compliance ruleset agreed per asset in Discovery.
`GET /v1/identities/{id}/eligibility`	Query, for a given token, whether a wallet may hold or receive it, and if not, why. Used by DataVault's platform to block a trade in the user interface before it reaches the chain.
`POST /v1/tokens/{id}/redemption-period`	Set, open, or close a redemption period.
`GET /v1/tokens/{id}/redemption-period`	Current period state.

Events back to DataVault

Every write returns the on-chain transaction hash, so DataVault's platform can confirm state against the chain rather than trusting the API response. In addition, RubySDK posts webhooks to an endpoint DataVault nominates, signed so DataVault's API layer can verify origin:

- `token.deployed` with contract address and chain
- `identity.registered` and `identity.revoked`

- `transfer.rejected` with the compliance reason code
- `redemption.period.opened` and `redemption.period.closed`
- `redemption.executed` with holder identity, quantity, and reference

Where DataVault would rather poll than receive webhooks, every event has a matching read endpoint. Authentication is by signed API key over mutual TLS. Writes carry an idempotency key, so a retried call after a network timeout does not produce a second on-chain transaction.

What a first integration honestly needs

Three of these calls carry the phase one use case: deploy a token, register a holder, query eligibility. Redemption periods are not needed until the first mine is producing. A working integration against our sandbox is a small piece of work once the specification is agreed. We estimate days rather than weeks, though the effort on DataVault's side depends on the shape of their API layer and is confirmed in Discovery. The work that takes time is not the API surface. It is agreeing the compliance ruleset per asset, agreeing the claim format with DataVault's KYC provider, and testing rejection behaviour end to end.

5 Security controls

The registry is the record of who owns a security. The controls over it are therefore part of the deliverable, not an operational detail settled later. This section sets out the architecture RubySDK commits to. Where a control depends on DataVault's own key management, signing arrangements, or incident process, that is stated and settled in Discovery during weeks 1 to 2 rather than specified here against an assumption about DataVault's environment.

Keys in the architecture

Key	Held by	Purpose and storage
Deployer	RubySDK	Deploys contracts only. Holds no standing authority over a token once deployed: ownership transfers to DataVault's owner key at the end of deployment. Under this engagement this key will be held in dedicated key-management hardware.
Owner	DataVault	Highest authority on the token contract: appoints and removes agents, sets the compliance modules, pauses the token. Multi-signature, with signer set and threshold agreed in Discovery. Not held by RubySDK.
Registrar agent	RubySDK, acting on DataVault's instruction	Writes to the identity registry: registers holders, updates claims, revokes. Scoped so it cannot move tokens, freeze balances, or alter compliance rules. Under this engagement this key will be held in dedicated key-management hardware, per environment, rotated on schedule.
Claim issuer	DataVault or DataVault's KYC provider	Signs the verification claims the registry trusts. Never held by RubySDK. This is what keeps the assertion that a holder passed verification inside DataVault's control.
Token agent	DataVault	Freeze, force transfer, and recovery. Held by DataVault, separate from both the owner key and the registrar key.

Separation of authority

ERC-3643 exposes these powers as distinct roles, and the deployment keeps them distinct. Registering a holder, revoking a holder, freezing an account, forcing a transfer, and pausing the token are five different authorities held by four different keys. No single key can both admit a holder to the registry and move that holder's tokens. The registrar key RubySDK operates is the narrowest of them: it writes registry entries and nothing else, so a compromise of RubySDK's key cannot move a DataVault client's asset.

Freeze and force transfer

These two powers can act on an asset the holder controls, so they are treated differently from the rest. Both sit on the token agent role, held by DataVault, not by RubySDK. Neither can be invoked through the RubySDK API, in this or any later phase: they are exercised by DataVault directly against the contract, and RubySDK does not hold the key that exercises them, so the authority and the audit sit with the party that carries the regulatory obligation. Every

invocation emits an event carrying the acting address, the affected holder, the amount, and the block, which makes each use permanently attributable on chain. The authorisation process that precedes an invocation, meaning who at DataVault may request a freeze and what evidence is required, is DataVault's to define; Discovery records it so the contract roles match the policy rather than the reverse.

Audit trail and independent verification

Every state-changing call returns the on-chain transaction hash. Registry writes, claim updates, revocations, deployments, redemptions, and rejected transfers all emit events. DataVault can reconstruct the full history of the registry from the chain alone, using its existing chain-publishing infrastructure, without querying RubySDK and without trusting a RubySDK-held log. That is the point of putting the register on chain: verification does not depend on us, and a discrepancy between our API and the chain resolves in favour of the chain.

Identity data

No personal data goes on chain. KYC is performed by DataVault or its provider, and the underlying records stay there. What is written on chain is a claim: a topic identifier, the issuer's address, and the issuer's signature, attached to the holder's on-chain identity contract. It asserts that the wallet behind that identity satisfied a named verification requirement, and it carries an expiry. It contains no name, document, date of birth, address, or account reference, and it is not a hash of the personal data from which the data could be confirmed by guessing at candidates. RubySDK does not receive, store, or process DataVault's customer records at any point in the integration.

Key compromise and recovery

The mechanisms that exist today:

- **Agent revocation.** The owner key removes a compromised agent, including RubySDK's registrar key, in a single transaction. Registry writes from that key stop at that block, with no dependency on RubySDK acting.
- **Registry integrity after compromise.** Registry state is reconstructible from chain events, so entries written by a compromised key can be identified by block range and corrected by the replacement key rather than requiring redeployment.
- **Holder key loss.** The recovery function moves a holder's balance to a replacement wallet bound to the same on-chain identity, invoked by DataVault's token agent role.
- **Pause.** The owner key halts transfers on the token while an incident is assessed.
- **Rotation.** Agent and registrar keys are rotated on a fixed schedule and on any suspicion of compromise, by appointing the new key and revoking the old.

What is not written yet is a joint incident runbook: detection thresholds, notification times between the two organisations, named contacts on each side, and the decision to pause. We are not going to claim one exists. Producing it is a Discovery deliverable in weeks 1 to 2, written against DataVault's existing incident process rather than imposed on it, and signed off before production go-live in week 8.

Signature schemes and post-quantum migration

Two layers of signing sit in this architecture and they migrate differently. Transaction signing is a property of the chain: the registry design does not depend on which scheme sits underneath it, so a chain-level migration to a new scheme does not require the registry to be rebuilt. Claim signing sits in the application layer, inside the architecture described above: the trusted issuer signs a claim and the holder's identity contract verifies that signature on chain. That is where the migration question actually bites. The answer is that claims carry expiries and are re-signed on renewal, so the claim layer migrates through ordinary operation rather than through a rebuild, and renewal is the migration path. Key material held in dedicated key-management hardware can be regenerated under new schemes as part of scheduled rotation. RubySDK tracks post-quantum developments and will support new schemes when the chain and the issuer tooling support them, rather than deploying them now.

API surface

Authentication, transport, and replay protection for the API are described in section 4: signed API keys over mutual TLS, with idempotency keys on writes so a retried call cannot produce a second on-chain transaction. Endpoint scoping follows the key separation above, so the credential DataVault's API layer uses for registry writes cannot reach a function it does not need.

RubySDK does not hold DataVault assets in the phase 1 architecture. Any custody arrangement would be scoped separately, with the insurance certificate provided for counsel's review before any assets are held.

> **Independent audit.** The contracts deployed for DataVault are put through an external security audit at week 4, run against the testnet deployment so there are weeks left to act on the findings before production. The report goes to DataVault rather than a summary of it. Any custom code written against DataVault's redemption period rules is in scope for that audit. Remediation of findings in RubySDK-authored code is included in the fixed fee; findings arising from DataVault-supplied logic are addressed by agreement.

6 Privacy and confidentiality

Phase one does not require a confidential chain. The registry, the compliance modules, and the token can be deployed on the EVM chain DataVault nominates, and that is the fastest route to a live registry inside 60 days.

Confidentiality becomes a requirement at institutional scale, when counterparties will not accept that their positions, holdings, and settlement patterns are readable by anyone with a block explorer. RubySDK's existing work on that problem carries forward from the earlier proposals:

- **Oasis Sapphire** is a confidential runtime that executes Solidity inside trusted execution environments. ERC-3643 deploys on it in its standard form, so the registry and the compliance modules run unchanged with contract state encrypted. Sapphire requires developers to invoke confidential execution explicitly, which is an architecture standard and an audit item rather than a technical obstacle.
- **Secret Network** encrypts contract state by default at the protocol level and supports native selective disclosure. It is CosmWasm-based, so ERC-3643 does not run on it natively and the contract library would require a port from Solidity.

On the evidence to date, and given that DataVault's requirement is EVM and standards-based, Oasis Sapphire is the stronger candidate for confidential deployment. The formal selection is a Discovery output, made against DataVault's asset classes, counterparty types, and jurisdictional obligations rather than against our preference.

> **Canton L1: roadmap, not phase one.** Native deployment of the RubySDK contract library on Canton, a settlement network built for bilateral confidentiality between institutional counterparties, is on RubySDK's roadmap. It is not available today, it is not a dependency of anything in this proposal, and nothing in the 60 day plan waits on it.

7 Phased delivery plan

Phase 1 is scoped to deliver within 60 days of signature. That is RubySDK's commitment, not an estimate. Phase 2 runs from day 60 to day 120. Phase 3 follows phase 2. The week structure below is what makes the 60 day target real rather than asserted, and it names what RubySDK delivers and what DataVault supplies at each step.

Phase 1: registry live, one mine tokenised end to end. Weeks 0 to 8.

Week	RubySDK delivers	DataVault supplies
Week 0	Agreement signed. Engineering kickoff within five working days, our engineers and DataVault's in the same room or the same call, no account managers in between.	Named engineering lead and the engineers who will do the work.
Weeks 1 to 2	Discovery, run concurrently with environment setup rather than ahead of it. Output at end of week 2: integration specification, compliance ruleset per asset class, chain selection for phase one, and a fixed-price Scope of Works for phase two.	Any existing redemption logic and the commercial rules governing redemption. API expectations for the integration. KYC provider name and claim format. Test environment access.
Weeks 3 to 4	Identity registry, compliance modules, token contract, and the redemption period module deployed to testnet. Sandbox API live. First deployment and holder-registration calls working from DataVault's API layer. External security audit run against the testnet deployment at week 4, with the report issued to DataVault.	Integration of their API layer against the sandbox. Test wallet set and sample holder records.
Weeks 5 to 6	Audit findings remediated and re-checked. One mine tokenised end to end on testnet: token deployed, holders registered, compliant transfer settled, non-compliant transfer rejected with its reason code, webhooks delivered to DataVault's API layer.	The first mine's asset data, holder list, and compliance rules. A decision on any finding that arises from DataVault-supplied redemption logic. Sign-off on rejection behaviour as it appears in the DataVault user interface.
Weeks 7 to 8	Production deployment. Registry live on mainnet. First mine tokenised on the production chain with a real approved-holder list.	Production KYC claim issuance. Go-live approval and the issuer key ceremony.

Day 60 lands at the end of week 8. What is live at that point is the registry, contracts deployable through the API, and one mine tradable with holders recorded and enforced on chain.

Phase 2: remaining assets and production redemption. Days 60 to 120.

- The remaining assets tokenised, each with its own compliance ruleset.
- Redemption periods in production, opened as each mine begins operating.
- Deployment across any additional chains DataVault nominates, from one registry.
- Issuer operations hardened: freeze, forced transfer, and lost-key recovery procedures documented and rehearsed with DataVault's operations team.

Phase 3: confidential and institutional settlement.

- Migration of the token library to the confidential chain selected in Discovery, if DataVault elects to take it.
- Canton L1 deployment when RubySDK's Canton library lands.

> **What puts 60 days at risk.** Two points, and neither is the engineering. The first and larger one is the week 1 to 2 inputs. If the redemption period rules, the KYC claim format, and test environment access arrive in week 1, the schedule holds. If they arrive in week 4, day 60 becomes day 80. We will say so in writing at the end of week 2 rather than at the end of week 8. The second is the audit at week 4. Findings in RubySDK-authored code are remediated inside weeks 5 to 6 and do not move the date. Findings in DataVault-supplied redemption logic need a decision from DataVault on what changes, and the time that decision takes is the part of the schedule we do not control. We will report the findings and the affected dates in the week the report lands.

8 What this proposal assumes

The commitments in this proposal are RubySDK's own and hold as written. The items below are what the proposal takes as given about DataVault's side. Each is confirmed in Discovery during weeks 1 to 2, and any that turns out to be wrong is reported in writing at the end of week 2 with its effect on the schedule.

- DataVault's platform exposes an API layer through which third-party services integrate over JSON and HTTPS.
- DataVault, or its KYC provider, performs identity verification and signs the verification claims the registry trusts.
- DataVault nominates the chain for the first production deployment, from the chains it publishes to, or a chain nominated for this deployment.
- DataVault defines the compliance rules per asset: holder caps, jurisdiction restrictions, lock-ups, and holding sizes.
- Where DataVault holds existing redemption logic, it is available to us as the input specification. Where it does not, we write the module to DataVault's commercial rules.
- DataVault holds and operates the owner key and the token agent key, with the signer set agreed before the key ceremony.
- DataVault provides a test environment and sandbox credentials for its API layer.
- DataVault retains the exchange, the client relationships, the assets, the tokenomics, the metadata storage and analytics, and the chain publishing.
- The instruments are treated as securities on DataVault's own determination, which RubySDK does not advise on.

9 What we need from DataVault to start

Item	Detail
Redemption rules and any existing logic	The commercial rules governing when a redemption period opens, how long it stays open, and what a holder receives, together with any existing contract code DataVault holds for it.
API expectations	The shape of the calls DataVault expects to make, so the API matches the platform's existing conventions rather than ours.
KYC provider and claim format	Which provider performs verification, what it returns, and who signs the claim as trusted issuer.
Test environment access	Sandbox credentials for the API layer and a test wallet set.
Chain targets	The chains DataVault currently publishes to, with the one selected for the first production deployment.
First mine	Which asset goes first, its asset data, holder list, and the jurisdictions its holders sit in.
Compliance rules per asset class	Holder caps, jurisdiction restrictions, lock-ups, and any minimum or maximum holding size.
Owner key	A multi-signature owner key, with the signer set and the threshold agreed before the key ceremony in weeks 7 to 8.
Token agent key	A separate key held by DataVault for freeze, force transfer, and recovery, distinct from the owner key.
Direct contract tooling	The tooling on DataVault's side to invoke the token contract directly for the functions that are not exposed through the RubySDK API, meaning freeze and force transfer.
Named engineering lead	The person our engineers work with daily from week 0.

10 Commercial terms

The structure is unchanged from the proposal issued on 18 May 2026: a fixed-fee first phase, a fixed price for the build quoted against a confirmed Scope of Works, and per-operation fees thereafter.

£125,000

Phase 1, fixed fee. Discovery, integration specification, registry and compliance modules deployed, one mine tokenised end to end, production go-live. Weeks 0 to 8.

Payable in full on signature.

Phase 1 as scoped here is larger than the Discovery phase priced in the earlier proposals. Discovery there was a four to six week study that ended in a specification. Here it is compressed into weeks 1 to 2 and runs alongside build, so that what DataVault holds at day 60 is a live registry rather than a document describing one. The fee is unchanged.

Phase	Fee	Basis
Phase 1 Registry and first asset	£125,000	Fixed fee, payable in full on signature. Deliverables as set out in section 7.
Phase 2 Remaining assets and redemption	Fixed price, quoted end of week 2	Priced against the Scope of Works produced in Discovery, so DataVault sees the number before phase 1 completes and can budget against it.
Phase 3 Confidential and Canton	Fixed price, quoted following phase 2	Scoped only if DataVault elects to take it. Not a commitment made now.

Ongoing fees

After production deployment there are two ongoing streams: an annual technology licence for platform access, and per-operation fees collected on chain at the point of execution. DataVault pays network gas directly to the chain. RubySDK takes no share of gas and no basis-point fee on transaction value.

Operation	RubySDK fee	Note
Token mint	$1.00	Per mint, collected on chain.
On-chain transfer	$0.01	Per transfer.
Metadata update	$0.25	Per update.
Escrow creation	$1.00	Where escrow is used.
Escrow settlement	$1.00	Where escrow is used.
Identity registration, claim update, revocation, redemption	Priced in the phase 2 Scope of Works	Registry operations were not in the earlier fee schedule. They will be priced at the same order of magnitude and confirmed in writing at end of week 2, before DataVault is committed to them.

Metadata update, escrow creation, and escrow settlement are carried forward from the May 2026 schedule and are priced for when DataVault elects to use them.

Annual technology licence: to be agreed, covering platform access across DataVault's exchanges. **Exchange launch fee:** a one-off deployment fee for each additional DataVault exchange platform, to be agreed.

11 Next step

This proposal is written to be handed to engineers and turned into a project without a further round of documents. The sequence that achieves that:

1. **Agreement signed.** Phase 1 scope as set out in section 7, at the fee in section 10.
2. **Engineering kickoff within five working days of signature.** DataVault's engineers and RubySDK's engineers, working directly. The commercial parties step out of the flow at that point.
3. **Week 1 inputs delivered** per section 9, so the week 2 specification and the phase 2 price are produced on schedule.

Everything else in this document is detail that engineering can settle between them once the agreement is in place.

RubySDK operates solely as a technology provider. RubySDK assumes no regulatory burden, licensing obligation, or compliance responsibility in connection with any DataVault AI exchange, tokenisation activity, or real-world asset transaction. All regulatory, legal, and compliance obligations remain exclusively with DataVault AI Inc. and its respective exchange entities. This proposal is confidential and is provided for the purposes of the parties' evaluation only.

Next step

Agreement signed, then our engineers and yours in the same room within five working days. The registry live at day 60.

Mark Abbott
Co-Founder, RubySDK
+44 7467 259590
rubysdk.com
rubytech.llc

Confidential. July 2026. © 2026 RubySDK. All rights reserved.