MINTED PROTOCOL · CONFIDENTIAL TECHNICAL DOCUMENTATION

PROTOCOL DEEP DIVE · V1.0

mUSD Protocol Architectural Disclosure.

A complete walkthrough of the Minted Protocol Canton-native implementation: every Daml module, every contract template, the security and compliance model, and the full set of governance and audit primitives.

NETWORK	MODULES	TEMPLATES	AUDIENCE
Canton · DAML 3.4.10	20+ Daml production modules	50+ Contract types	Institutional Reviewer

01 Protocol Overview

Minted Protocol is a Canton-native institutional stablecoin protocol implementing two structurally separate products:

- **mUSD** — non-yield-bearing payment stablecoin, the Canton-native cash leg for institutional settlement
- **smUSD** — structurally separate Institutional Yield Vault, ERC-4626-equivalent share-price model implemented natively on Canton, accessing curated yield sleeves

This document is the result of a code-level walkthrough of the canonical Canton-side repository. It documents every Daml module, every contract template, the security model, and the governance and audit primitives that make up the protocol.

ARCHITECTURAL POSTURE

The protocol is implemented entirely on Canton Network using Daml 3.4.10 with the Daml Finance asset model. Every protocol contract is a Canton-native Daml template — per-owner, per-custodian discrete contracts with their own signatories. Every state-changing transaction commits atomically across separately controlled participant nodes via Canton's atomic commit protocol with sub-transaction privacy.

From the institutional perspective, all user-facing transactions execute under Canton's privacy and DAML signatory model. mUSD is a Canton-native asset; smUSD vault interests are Canton-native; lending positions, governance proposals, audit receipts, and compliance enforcement all operate within the Canton ledger with no public-chain DeFi dependency in the user-facing path.

REPOSITORY LAYOUT

PATH	CONTENTS	LINES (APPROX)
`/daml/`	Canton-side protocol contracts (Daml 3.4.10)	~10,000
`/daml/Minted/Protocol/V3.daml`	Unified protocol module (consolidated templates)	2,027
`/daml/CantonLending.daml`	Largest single Daml module — 9 templates, full lending protocol	2,304
`/daml/InitProtocol.daml`	Idempotent initialization script (Canton 3.4+ compatible)	174

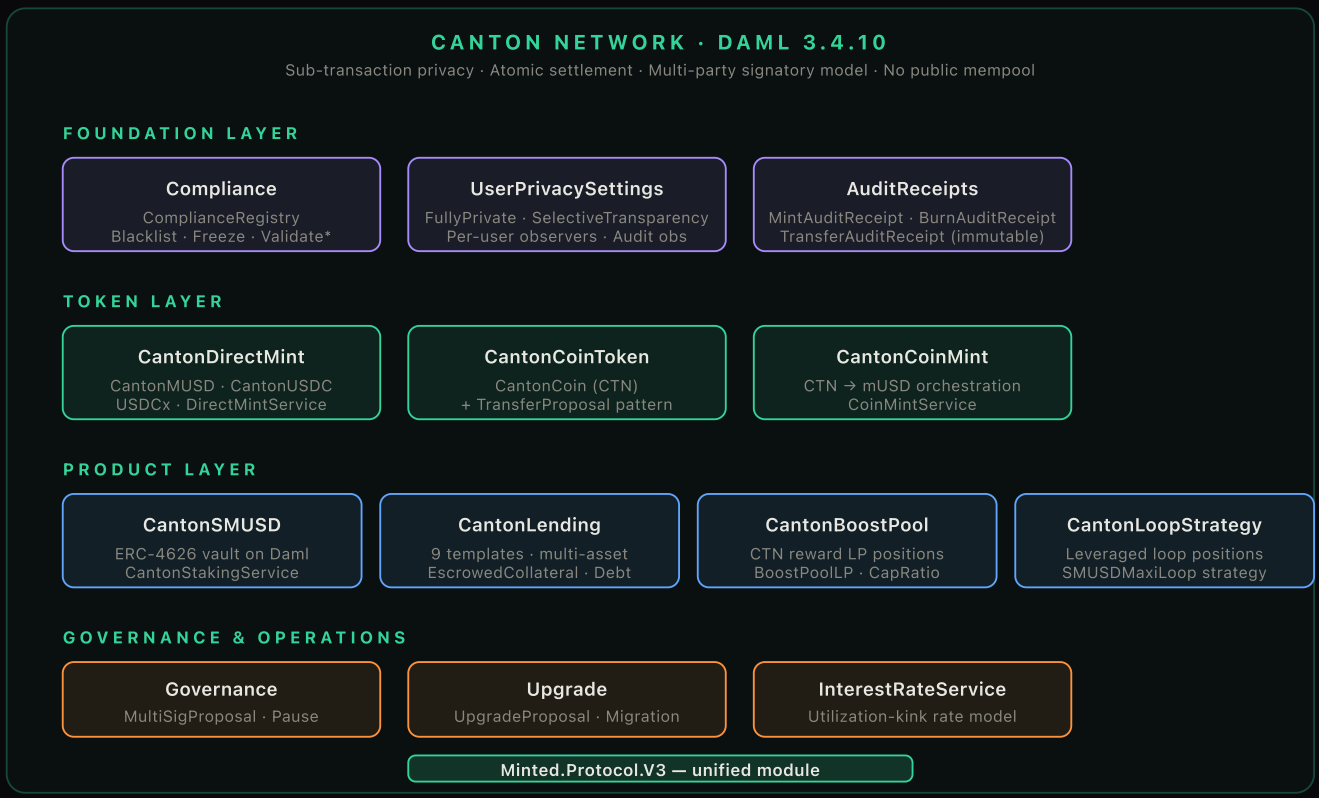
DOCUMENT SCOPE

This document covers the Canton-native protocol surface end-to-end. The protocol's external settlement and yield-source counterparties (T-RIZE Kairos sleeve, Obligate RWA10 sleeve, Fasanara sleeve) are documented separately in the smUSD Vault Economic Terms and the Institutional Flow Memo. The bridging mechanics that allow off-Canton settlement counterparties to interoperate with the Canton-native protocol are out of scope for this disclosure and are documented separately.

02 System Architecture

The protocol consists of approximately 20 production Daml modules organized in three layers: the foundation layer (compliance, privacy, audit), the token layer (mUSD, USDCx, CTN, vault shares), and the product layer (lending, yield vault, boost pool, leveraged loop, governance, upgrade). All modules ultimately consolidate into the unified `Minted.Protocol.V3` module that the frontend and API layer expect.

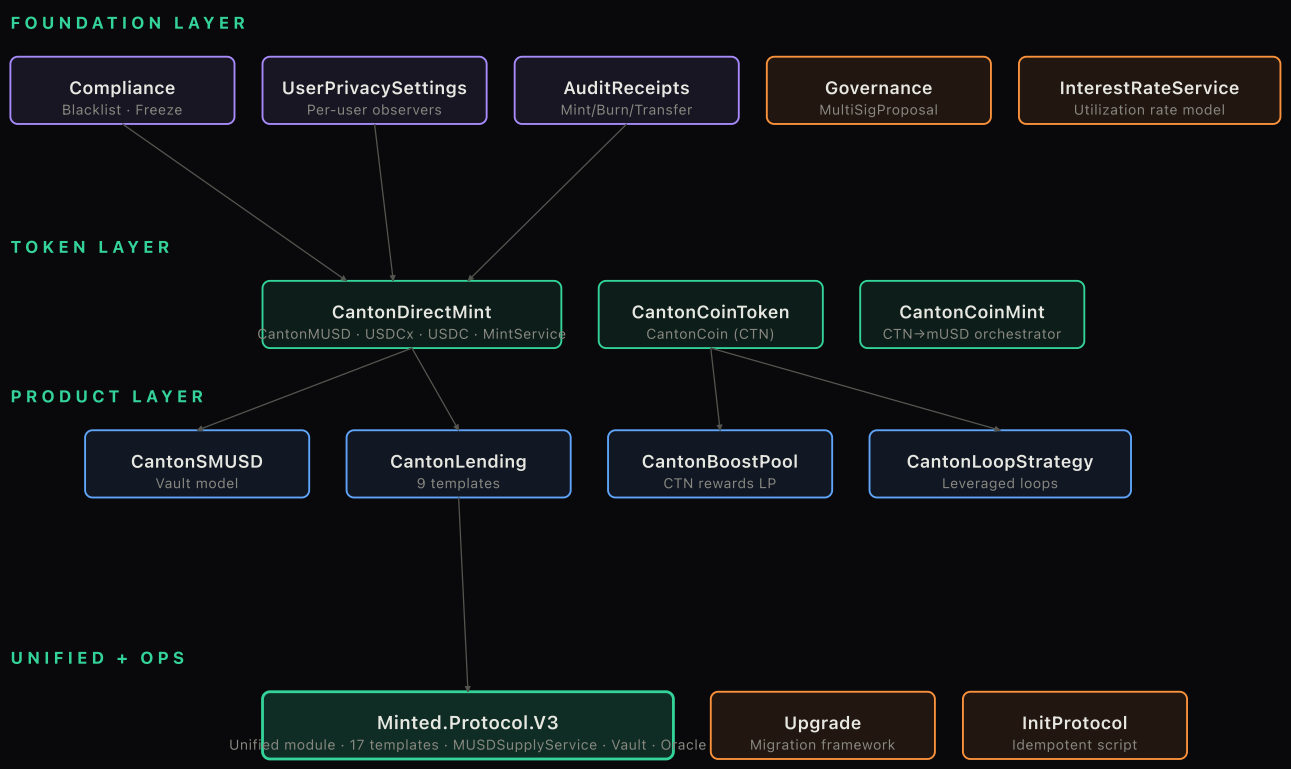
FIGURE 1 — CANTON-NATIVE MODULE ARCHITECTURE



03 Daml Module Dependency Graph

Every product module imports from the foundation layer (Compliance, UserPrivacySettings, AuditReceipts) for compliance enforcement, observer derivation, and audit trail generation. The token layer (CantonDirectMint, CantonCoinToken) is imported by every product module; the unified `Minted.Protocol.V3` module consolidates the full surface for frontend / API consumption.

FIGURE 2 — DAML MODULE DEPENDENCIES



04 Foundation Modules

Three foundation modules establish the cross-cutting concerns used by every product module: compliance enforcement, privacy controls, and audit trail.

Compliance.daml

Single template `ComplianceRegistry`, signatory-controlled by the regulator party with the operator as observer. The registry maintains `Set Party` structures for blacklisted and frozen parties (O(log n) lookup).

CHOICE	CONTROLLER	EFFECT
`BlacklistUser`	regulator	Add party to blacklist with audit reason
`RemoveFromBlacklist`	regulator	Remove party with explicit audit reason (no expiry)
`FreezeUser` / `UnfreezeUser`	regulator	Per-party asset freeze (frozen can receive, cannot send)
`ValidateMint`	operator	Nonconsuming — minter must not be blacklisted
`ValidateTransfer`	operator	Nonconsuming — both parties checked, sender freeze-checked
`ValidateRedemption`	operator	Nonconsuming — redeemer not blacklisted or frozen
`BulkBlacklist`	regulator	Up to 1000 parties per call (mass-freeze emergency capacity)

Architectural note: the `Validate*` choices are nonconsuming so they can be called from within other templates' choices without archiving the registry. Controller is the operator (not the regulator) so cross-template compatibility holds; the regulator's policy is enforced through the data on the contract, which only the regulator can mutate.

UserPrivacySettings.daml

Canton is private by default: only signatories see contract state. UserPrivacySettings is the unified privacy toggle that lets users opt into selective transparency (add specific observers like compliance, fund admin, auditor) per their institutional needs.

CHOICE	CONTROLLER	EFFECT
`Privacy_AddObserver`	user	Add observer to all future contracts (transitions to SelectiveTransparency)
`Privacy_RemoveObserver`	user	Remove observer; transitions to FullyPrivate if last
`Privacy_GoPrivate`	user	Remove all observers
`Privacy_SetObservers`	user	Bulk update with labels (e.g., "Auditor", "Compliance")

The helper function `lookupUserObservers` is called from every product module's create flow. **DAML-H-02 fix:** in LF 2.x contract keys are no longer guaranteed unique, so lookupByKey was removed; users now pass an Optional ContractId explicitly.

AuditReceipts.daml

Three immutable receipt templates produce permanent ledger evidence of every state-changing action. Receipts are never archived in normal operation.

TEMPLATE	CREATED WHEN	CAPTURES
`MintAuditReceipt`	mUSD minted via DirectMint	minter, amount, fee, supplyAfter, txHash

`BurnAuditReceipt`	mUSD burned/redeemed	burner, amount, fee, netRedeemed, supplyAfter
`TransferAuditReceipt`	token transfer (any of 5 token types)	sender, recipient, tokenType, amount, txHash

Each receipt has signatory `operator` with the regulator and the user as observers. The receipt contracts give regulators a queryable ledger trail without needing to scan archived state.

05 Token Layer

The token layer defines the protocol's core asset templates. Every token follows the same pattern: signatory `(issuer, owner)`, observer `privacyObservers`, transfer-proposal pattern for safe dual-signatory transfers, and mandatory compliance validation at transfer initiation.

CantonDirectMint.daml — Asset Templates

TEMPLATE	PURPOSE
`CantonUSDC`	USDC deposited on Canton (entry asset for direct mint path)
`USDCx`	xReserve-bridged USDC on Canton (Circle CCTP-attested, sourceChain + cctpNonce fields)
`CantonMUSD`	Canton-native mUSD with embedded MPA hash and URI
`CantonDirectMintService`	The minting orchestrator (rate limits, supply cap, fees, MPA reference)
`RedemptionRequest`	Created on burn — fulfilled by the redemption-settlement pipeline
`ReserveTracker`	Cumulative deposit/redeem counters for accounting reconciliation

CANTONMUSD INVARIANTS

The CantonMUSD template enforces the following at the type level:

```
template CantonMUSD
  with
    issuer : Party
    owner : Party
    amount : Money          -- Numeric 18 precision
    agreementHash : Text    -- SHA-256 hash of MPA PDF
    agreementUri : Text     -- URI to legal terms
    privacyObservers : [Party]
  where
    signatory issuer, owner
    observer privacyObservers

    ensure amount > 0.0
      && DA.Text.length agreementHash == 64
      && DA.Text.length agreementUri > 0
```

Every token instance carries the Master Participation Agreement reference inline — users cannot hold mUSD without acknowledging the MPA via signatory consent.

CANTONDIRECTMINTSERVICE — MINT CHOICES

The service supports three entry asset types:

CHOICE	ENTRY ASSET	EFFECT
`DirectMint_Mint`	CantonUSDC	Direct USDC deposit → mint mUSD net of fee
`DirectMint_MintWithUSDCx`	USDCx (xReserve)	USDCx-equivalent → mint mUSD net of fee
`DirectMint_MintForCoin`	CantonCoin	CTN at oracle price → mint mUSD; operator absorbs CTN exposure
`DirectMint_Redeem`	(burn mUSD)	Burn mUSD → create RedemptionRequest

`DirectMint_RedeemFromInventory`	(operator inventory)	CIP-56-native redeem path using operator-owned inventory tokens

RATE LIMITING + SUPPLY CAP

The service enforces three quantitative controls inline on every mint/redeem:

- **Per-transaction bounds:** `minAmount ≤ usdc.amount ≤ maxAmount`
- **Supply cap:** `currentSupply + netAmount ≤ supplyCap`
- **24-hour rolling rate limit:** net mint volume in current 24h window cannot exceed `dailyMintLimit`; burns offset mints in the window

The 24h window is implemented as a soft reset: if `now − lastRateLimitReset ≥ 24h`, counters reset before checking. This avoids rigid daily UTC boundaries.

CantonCoinToken.daml — CantonCoin (CTN)

Standalone module extracted from CantonBoostPool to break a cyclic dependency. The `CantonCoin` template wraps the native Canton validator token with the same signatory/observer/transfer-proposal pattern as the other tokens, plus mandatory compliance validation on transfer.

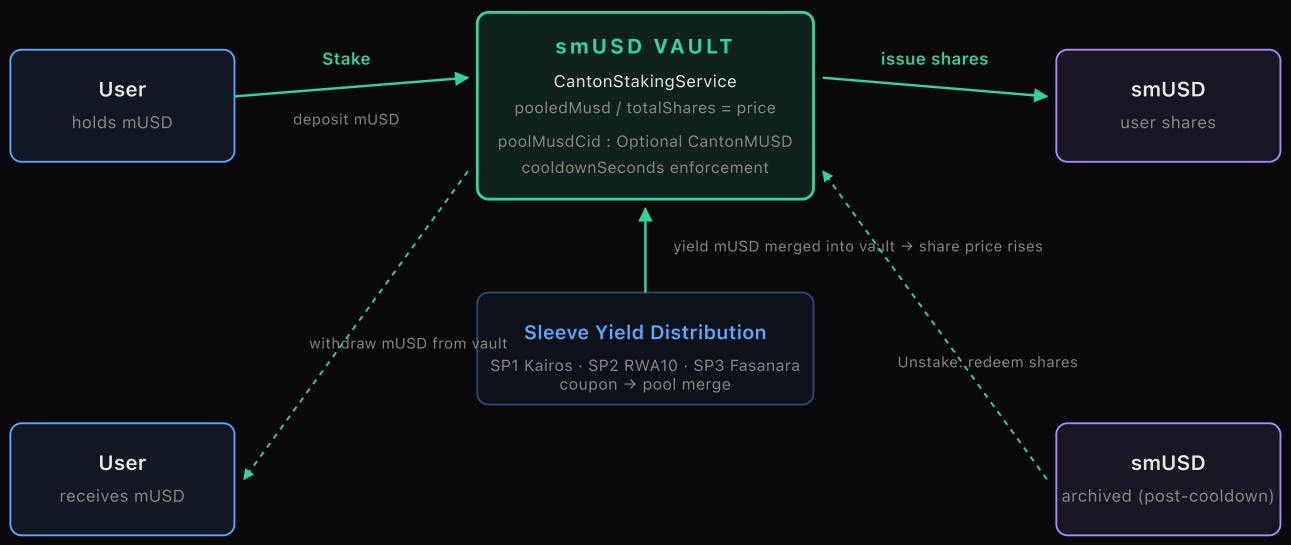
CantonCoinMint.daml — CTN to mUSD Orchestrator

The `CoinMintService` template orchestrates a multi-step flow: user has CTN, operator has USDCx, the choice atomically (1) burns the CTN, (2) transfers operator's USDCx to user at oracle price, (3) calls `DirectMint_MintWithUSDCx` on the user's behalf. This unifies the user experience across the three accepted entry-asset paths.

06 smUSD Yield Vault

The smUSD vault is implemented in `CantonSMUSD.daml` as a Daml-native equivalent of the ERC-4626 tokenized vault standard. The vault holds mUSD on stake (rather than burning it), tracks user shares against the pool, and grows the share price as yield is distributed into the vault from the smUSD sleeve sources.

FIGURE 3 — SMUSD VAULT STAKE / UNSTAKE CYCLE



Templates & Choices

TEMPLATE	PURPOSE
`CantonSMUSD`	User position (shares, entry price, stakedAt timestamp for cooldown)
`CantonSMUSDTransferProposal`	Dual-signatory transfer pattern (DL-C2 fix)
`CantonStakingService`	Vault accounting orchestrator with poolMusdCid backing

VAULT MECHANICS

- **Vault holds the mUSD.** On stake, the user's mUSD contract is archived and a new operator-owned CantonMUSD is created and merged into the existing pool contract. mUSD is not burned — it accumulates as TVL.
- **Share-price model.** `localSharePrice = pooledMusd / totalShares`. New shares issued = `deposit / localSharePrice`. Mirrors ERC-4626 deposit() semantics.
- **Yield merges into pool.** Sleeve coupons and maturity proceeds are routed to the operator-owned pool contract; `pooledMusd` increases; `localSharePrice` rises automatically.
- **Cooldown enforcement (D-M01 fix).** Each smUSD position records `stakedAt : Time`. Unstake asserts `now - stakedAt ≥ cooldownSeconds`.
- **Compliance enforced at every touch point.** Stake calls `ValidateMint`; transfer calls `ValidateTransfer`; unstake calls `ValidateRedemption`.

STAKEFROMINVENTORY (CIP-56-NATIVE PATH)

Alternative stake path where the user's CIP-56 tokens are archived at the API layer and operator inventory tokens are consumed atomically. Same economic invariants as `Stake` but eliminates the intermediate user-owned CantonMUSD requirement — cleaner flow for operator-mediated CIP-56 onboarding.

07 CantonLending — Overcollateralized Lending Protocol

`CantonLending.daml` is the largest single module (2,304 lines, 9 templates). It implements an overcollateralized lending market with multi-asset collateral acceptance, dynamic interest rates, escrowed collateral positions, and a liquidation engine.

Templates

TEMPLATE	ROLE
`CantonPriceFeed`	Asset price oracle with provider-signed updates and emergency override
`EscrowedCollateral`	User collateral position (locked, not transferred); supports add/withdraw/seize
`CantonDebtPosition`	User borrow position; tracks principal, interest, accrual time, current rate
`LendingCollateralAggregate`	Per-asset total collateral aggregator
`CantonSupplyPosition`	User supply position (lender side); tracks shares, accrued interest
`LendingSupplyAggregate`	Per-asset total supply aggregator
`LendingAggregate`	Total borrow aggregator (used for utilization calculation)
`CantonLendingService`	Orchestrator; entry point for all deposit / borrow / repay / liquidate flows
`CantonLiquidationReceipt`	Immutable liquidation audit record

Accepted Collateral Types

The lending service accepts five collateral types via dedicated deposit choices:

COLLATERAL TYPE	CHOICE	SOURCE MODULE
CantonCoin (CTN)	`Lending_DepositCTN`	CantonCoinToken
USDC	`Lending_DepositUSDC`	CantonDirectMint
USDCx	`Lending_DepositUSDCx`	CantonDirectMint
smUSD	`Lending_DepositSMUSD`	CantonSMUSD

Lifecycle

Borrower flow

1. Deposit collateral → EscrowedCollateral
2. Borrow mUSD against escrow at LTV
3. Accrue interest at dynamic rate
4. Repay debt or face liquidation

Supplier flow

1. Supply mUSD into pool
2. Receive CantonSupplyPosition
3. Earn interest from borrowers
4. Withdraw partial or full at any time (subject to utilization)

Liquidation flow

1. Health factor monitored: `collateral × LTV / debt`
2. Below threshold → liquidation eligible
3. Keeper repays debt portion (close factor) for collateral + bonus

4. CantonLiquidationReceipt emitted

PER-ASSET CONFIGURATION (COLLATERALCONFIG)

Each accepted collateral asset has its own configuration:

- **liquidationThreshold** — health-factor threshold, e.g., 1.5 (150%)
- **interestRateBps** — base rate; modulated by InterestRateService utilization model
- **liquidationPenaltyBps** — protocol fee on liquidations
- **liquidationBonusBps** — keeper incentive bonus
- **closeFactorBps** — max % of debt liquidatable per call (e.g., 5000 = 50%)
- **dustThreshold** — below this debt size, force full liquidation to prevent dust positions

08 BoostPool · LoopStrategy

CantonBoostPool.daml — CTN Reward Distribution

Distributes Canton Coin rewards (earned via Featured Application status on Canton Network) to LP positions. Templates:

- `BoostPoolLP` — LP token with deposit record
- `BoostPoolDepositRecord` — per-asset deposit accounting
- `CantonBoostPoolService` — orchestrator with cap ratio, fees, reward distribution

Key choices: `Deposit`, `Withdraw`, `DistributeRewards`, `SyncCantonPrice`, `WithdrawProtocolRewards`, `WithdrawProtocolFees`. Integrates with EscrowedCollateral from CantonLending for collateral-tied LP positions.

The boost pool is the protocol's mechanism for sharing the Canton ecosystem rewards earned through Featured App status with participants who provide productive engagement (collateral lockup, supply provision, sleeve participation). Rewards accrue in CTN and are claimable through the standard withdrawal choice.

CantonLoopStrategy.daml — Leveraged Looping

Implements the SMUSDMaxiLoop strategy — repeatedly stake smUSD, deposit smUSD as collateral, borrow mUSD, restake. Each loop iteration increases effective yield exposure at the cost of increased liquidation risk.

TEMPLATE	PURPOSE
`CantonLoopPosition`	Active loop position with iteration count, total collateral, total debt
`CantonLoopStrategyService`	Orchestrator with entry / unwind / emergency-close choices
`CantonLoopStrategyConfig`	Governance-controlled parameters (max loops, target LTV, min health factor)
`CantonLoopRequest` / `CantonUnwindRequest`	Two-phase commit pattern for loop execution

Default LoopConfig: `maxLoops` (cap on iteration count), `targetLtv` (e.g., 75%), `ctnTargetLtv` (CTN-collateral specific), `minHealthFactor` (safety threshold above 1.0). Governance can tune via `LoopConfig_*` choices.

The strategy is positioned as an opt-in advanced product. Default protocol participants do not interact with leveraged loops; the module is admitted to the protocol surface for sophisticated counterparties who want amplified yield exposure under explicit risk acknowledgment.

09 Governance · Upgrade · Interest Rates

Governance.daml

Multi-signature governance framework with role-based access control, time-locked execution, and immutable action logs.

TEMPLATE	PURPOSE
`GovernanceConfig`	Single-instance config: governors with roles, standard/elevated thresholds, timelock duration, proposal expiry
`MultiSigProposal`	Generic proposal with M-of-N approval, payload + payloadHash, scoped targetModule
`MinterRegistry`	Authorized minters with per-minter quotas; replenished via governance proof
`GovernanceActionLog`	Immutable record of executed actions; consumed via ConsumeProof to prevent replay
`EmergencyPauseState`	Multi-guardian pause threshold; requires governance proof to resume

ACTION TYPES & THRESHOLDS

ACTION	THRESHOLD	USE CASE
ParameterUpdate	Standard	Fee changes, rate limits
MinterAuthorization	Standard	Add/remove authorized minters
SupplyCapChange	Standard	Modify global supply cap
EmergencyPause	Elevated	Pause protocol
ContractUpgrade	Elevated	Upgrade contracts
TreasuryWithdrawal	Standard	Withdraw from treasury
GovernorChange	Elevated	Modify governor set

PROPOSAL LIFECYCLE

1. Proposer creates `MultiSigProposal` with payload, hash, action type, expires at proposalExpiry
2. Governors call `Proposal_Approve`; on threshold, status → Approved with timelockEndsAt = now + timelockDuration
3. After timelock, executor calls `Proposal_Execute` which creates `GovernanceActionLog`
4. The downstream operation that requires the proof calls `ConsumeProof` to archive the log entry, preventing replay

D-H-01 fix: the GovernanceActionLog signatory was changed from `(operator, executedBy)` to operator-only because `archive governanceProofCid` in operator-controlled choices would fail when executedBy ≠ operator.

Upgrade.daml — Migration Framework

Templates: `UpgradeProposal`, `UpgradeRegistry`, `MigrationTicket`, `UpgradeMigrationLog`. Supports approve / activate / record / complete / emergency-rollback lifecycle. Each migration creates an immutable log entry for audit trail.

InterestRateService.daml — Rate Model

Canton-side interest rate calculator with utilization-based borrow rate (kink model), supply rate derivation, and split between borrowers and protocol reserves.

CHOICE	RETURNS	USE
`RateService_GetUtilization`	Int (bps)	Pool utilization snapshot
`RateService_GetBorrowRate`	Int (bps)	Current borrow APR
`RateService_GetSupplyRate`	Int (bps)	Current supply APY
`RateService_CalculateInterest`	Decimal	Interest accrual over time delta
`RateService_SplitInterest`	(Decimal, Decimal)	(borrower portion, reserve portion)

10 Minted.Protocol.V3 — Unified Module

`daml/Minted/Protocol/V3.daml` is a 2,027-line consolidated module that aggregates the protocol's full template surface in one place. The unified module is what `InitProtocol.daml` imports for protocol initialization, and what the frontend / API layer expects for choice naming convention (`TemplateName_ChoiceName`).

Templates in V3 (Canton-Native Surface)

#	TEMPLATE	FUNCTION
1	`CantonUSDC`	USDC deposit asset (V3 variant)
2	`CollateralDepositProof`	Audit proof of collateral acceptance
3	`MUSDSupplyService`	Supply cap orchestrator with large-mint approval queue
4	`MintedMUSD`	Canton mUSD with compliance enforcement
5	`MUSDTransferProposal`	Dual-signatory transfer pattern
6	`PriceOracle`	Provider-signed price feed
7	`LiquidityPool`	On-chain DEX for atomic operations
8	`Vault`	Collateralized debt position (CDP)
9	`VaultManager`	Factory for creating Vault instances; default config + allowed collaterals
10	`LiquidationReceipt`	Immutable per-liquidation audit trail
11	`LiquidationOrder`	Keeper coordination contract
12	`CantonDirectMint`	V3 variant of direct mint
13	`CantonSMUSD`	V3 variant of yield vault
14	`CooldownTicket`	Tracks stake time for withdrawal cooldown
15	`TicketTransferProposal`	Transfer-proposal pattern for cooldown tickets

The V3 module also contains settlement-boundary contracts that coordinate Canton-native settlement events with external counterparties (sleeve issuers, custody chains, redemption fulfillment partners). These templates are out of scope for this Canton-native deep dive and are documented separately in the smUSD Vault Economic Terms and the Institutional Flow Memo.

VaultManager Default Configuration

From `InitProtocol.daml`, the production VaultManager initializes with:

- `liquidationThreshold = 1.5` (150% collateralization required)
- `interestRateBps = 500` (5% APR base)
- `liquidationPenaltyBps = 500` (5% protocol fee)
- `liquidationBonusBps = 100` (1% keeper bonus)
- `closeFactorBps = 5000` (50% per liquidation call)

- `dustThreshold = 50.0` (force full liquidation below 50 mUSD debt)

11 Security Model

The protocol uses Daml's signatory model as the foundation of its security posture. Every state-changing action requires explicit multi-party authorization at the contract level — there is no notion of unilateral mutation of held assets.

DAML Signatory Discipline

Every protocol token uses `signatory issuer, owner`. Consequences:

- **Mint requires both operator and recipient authorization.** The operator alone cannot create a CantonMUSD to a non-consenting party.
- **Burn requires both signatories.** Even if operator keys are compromised, the operator cannot unilaterally burn user-held mUSD.
- **Transfer is two-step.** Owner initiates a TransferProposal (signatory: issuer + owner; observer: newOwner); recipient must accept via `*TransferProposal_Accept` to become signatory of the new contract instance. This prevents forced signatory obligations on unwitting recipients (DL-C2 fix).

Compliance Enforcement Surface

The ComplianceRegistry's `ValidateMint` / `ValidateTransfer` / `ValidateRedemption` nonconsuming choices are called inline at every protocol entry point:

PROTOCOL ACTION	COMPLIANCE CHECK CALLED
DirectMint_Mint / MintWithUSDCx / MintForCoin	ValidateMint(minter)
CantonMUSD_Transfer / USDCx_Transfer / CantonCoin_Transfer / CantonUSDC_Transfer / SMUSD_Transfer	ValidateTransfer(sender, receiver)
DirectMint_Redeem	ValidateRedemption(redeemer)
Stake / StakeFromInventory / Unstake	ValidateMint or ValidateRedemption
Lending_Deposit* / Lending_Borrow / Lending_Repay	ValidateMint or ValidateRedemption

Key-Compromise Resistance

The combination of Daml signatory discipline and DAML-H-01 compliance enforcement produces a meaningful key-compromise resistance property. If operator keys are stolen, the attacker cannot:

- Drain user-held mUSD or smUSD positions (signatory-controlled archives)
- Mint mUSD to attacker-controlled wallets (compliance check requires ValidateMint, attacker's address blacklisted via regulator)
- Modify the ComplianceRegistry (regulator-only signatory)
- Bypass governance thresholds on protected actions (multi-sig proposals required for parameter changes, supply caps, upgrades)

Audit Findings — Remediation Status

The codebase contains explicit code-level references to remediated audit findings. Sample fix tags found in production code:

TAG	SEVERITY	DESCRIPTION
HIGH-01	High	Mandatory compliance validation on smUSD transfers
HIGH-07	High	Governance co-signer requirement on critical staking ops
D-M01	Medium	stakedAt timestamp added for cooldown enforcement

D-M08	Medium	Validate sender + recipient against compliance on mUSD transfer
DL-C2	Critical	Transfer-proposal pattern (no forced signatory obligations)
DAML-H-01	High	Mandatory compliance on transfer proposals; archive proof after use
DAML-H-02	High	LF 2.x contract-key migration (lookupByKey replaced); executor authorization on ConsumeProof
DAML-H-04	High	Mandatory compliance hooks (cannot be bypassed)
DAML-M-02	Medium	Multi-guardian pause threshold
DAML-M-04	Medium	MinterRegistry quota enforcement (operator-controlled)
DAML-M-05	Medium	Compliance check on USDC transfer recipient
DAML-M-09	Medium	Timelock duration in proposal field (LF 2.x compatibility)
DAML-C-02	Critical	Immutable AuditReceipts on every mint/burn/transfer
D-H-01	High	GovernanceActionLog signatory restriction (operator only)
D-L-04	Low	Bulk blacklist cap raised to 1000 for emergency response
C-DAML-01	Critical	ConsumeProof one-time use
C-DAML-03	Critical	Scoped governance via targetModule field
AUDIT-C-01	Critical	Oracle-anchored pricing on LiquidityPool

Each tag corresponds to an external audit finding (Softstack mUSD Canton Protocol audit + Softstack Institutional Vault audit, both dated 28 February 2026). The presence of explicit fix tags in the code indicates traceable remediation discipline.

12 Operational Posture

Validation

- **Five North 5N Sandbox Validation:** 27 phases passed, 244 test cases, 99.3% overall coverage, 100% DAML coverage, 95.7% operations coverage, 116+ lifecycle exercises, 6-hour stability soak 72/72 samples passing, zero out-of-memory events
- **Full DvP settlement proven end-to-end** via the validation suite

Audits

AUDIT	DATE	CRITICAL	HIGH	MEDIUM	LOW
Softstack mUSD Canton Protocol	2026-02-28	0	5 / 5 remediated	8 / 12 remediated, 4 acknowledged	18 / 23 remediated, 5 acknowledged
Softstack Institutional Vault	2026-02-28	0	4 / 4 remediated	8 most remediated, small number acknowledged	11 most remediated

Acknowledged findings are disclosed in the published audit PDFs with documented compensating controls.

Validator Infrastructure

- **Five North SV, LLC** — Canton Super Validator
- DevNet, TestNet, MainNet validator nodes
- 99.9% uptime SLA
- Signed validator agreement, executed 2026-03-23

Compliance Stack

- FinCEN MSB registered
- BSA/AML, OFAC screening (Chainalysis + TRM Labs integration)
- Texture Capital signed LOI 2026-04-28 (FINRA BD/ATS/transfer agent)
- Reg D 506(c) / Reg S framework for U.S. distribution
- Wallace Glausi as General Counsel; Bob Feil as Chief Advisor (ex-VP FRB Dallas)

Initialization Posture (InitProtocol.daml)

Protocol initialization is idempotent: `getOrAllocateParty` looks up parties by hint prefix before allocation (Canton 3.4+ does not allow re-allocation); `query @Template` checks for existing instances before create. Re-running the script is safe and skips already-deployed contracts. Default Devnet initialization creates ComplianceRegistry, MUSDSupplyService (10M cap, 100K large-mint threshold), PriceOracle, VaultManager, and LiquidityPool.

13 Conclusion

The Minted Protocol implements a functionally complete Canton-native institutional stablecoin protocol. Twenty-plus production Daml modules implement compliance enforcement, two-token architecture (mUSD payment + smUSD vault), overcollateralized lending with multi-asset collateral, leveraged loop strategies, multi-signature governance, time-locked upgrades, and immutable audit trails.

The codebase exhibits the institutional-grade engineering signals that distinguish credible Canton protocols from prototype work:

- Daml signatory discipline applied consistently across all asset templates
- Compliance hooks wired into every state-changing entry point with no bypass paths
- Transfer-proposal pattern preventing forced signatory obligations
- Immutable audit-receipt creation on every mint, burn, and transfer
- Multi-signature governance with timelocks, role-based access, and proof-consumption discipline
- Two production-grade external audits (Softstack) with traceable in-code remediation tags
- Five North 5N validation at 99.3% coverage with full DvP proven end-to-end
- Idempotent protocol initialization compatible with Canton 3.4+ party-allocation constraints

The architecture takes full advantage of Canton's institutional-finance primitives: sub-transaction privacy, multi-party signatory authorization, atomic settlement, and per-contract observer control. These primitives are unavailable on public-chain DeFi rails and are the structural reason institutional counterparties operate on Canton rather than on global-state public ledgers.

This document is intended as a Canton-native code-level architectural disclosure suitable for institutional reviewer diligence. For implementation-level questions or full code review access, contact the engineering team via the standard data-room counterparty channel. External-counterparty settlement mechanics (sleeve managers, custody arrangements, redemption fulfillment partners) are documented separately in the smUSD Vault Economic Terms and the Institutional Flow Memo.

Disclaimer: This document reflects the Canton-native portion of the protocol implementation at the time of writing. Code may evolve between document publication and mainnet launch. Reviewers requiring a definitive snapshot should request a commit hash reference. This document is for institutional-reviewer diligence; it is not an offer to sell or a solicitation of an offer to buy any security or instrument. The mUSD whitepaper, smUSD vault economic terms, audit-and-validation summary, and Softstack audit PDFs are the canonical disclosure documents at deck.minted.app.