Custody Protocol — Complete Type Definitions, RocksDB Keys, and Workflow Mapping (V1)

This document is the consolidated, implementation-oriented specification for the custody-native protocol we designed:

- Full type definitions (SCALE-oriented, C++-mappable)
- RocksDB key generation methods (where applicable)
- The reason each type exists and what it controls
- Where each type participates in the custody workflow

Encoding model: SCALE canonical encoding (tagged variants + ordered structs). Storage model: deterministic RocksDB keyspace.

0) Core encoding + key-building interfaces

These interfaces are the *only* primitives every type relies on.

```
// SCALE codec interface (conceptual)
struct ScaleWriter {
  void write_u8(uint8_t);
  void write_u16_le(uint16_t);
  void write_u32_le(uint32_t);
  void write_u64_le(uint64_t);
  void write_u128_le(uint128_t);
  void write_bytes(span<const uint8_t>);               // raw
  void write_compact_u32(uint32_t);                    // for vec lengths
  template<typename T> void write_vec(const vector<T>&); // compact-len + T
  void write_vec_bytes(const vector<uint8_t>&);        // compact-len + bytes
};

struct ScaleReader {
  uint8_t  read_u8();
  uint16_t read_u16_le();
  uint32_t read_u32_le();
  uint64_t read_u64_le();
  uint128_t read_u128_le();
  uint32_t read_compact_u32();
  template<typename T> vector<T> read_vec();
  vector<uint8_t> read_vec_bytes();
};

template<typename T> void scale_encode(ScaleWriter&, const T&);
template<typename T> T scale_decode(ScaleReader&);

// RocksDB key builder
struct KeyBuilder {
  vector<uint8_t> b;
  KeyBuilder& lit(const char* ascii);                  // append literal bytes
  KeyBuilder& byte(uint8_t);
  KeyBuilder& raw(span<const uint8_t>);
  KeyBuilder& hash32(const array<uint8_t,32>&);
  KeyBuilder& u32_le(uint32_t);
  KeyBuilder& canon_signer(const SignerId&);           // see SignerId section
  vector<uint8_t> finish();
};
```

1) Common primitives

```
using Hash32 = array<uint8_t, 32>;
using Bytes = vector<uint8_t>;
using Amount = uint128_t;            // or boost::multiprecision::uint128_t
using TimestampMs = uint64_t;
using DurationMs = uint64_t;
```

2) Identity and transaction envelope

SignerId

Purpose / why it exists: Defines the acting authority (who can propose/approve/execute/admin/attest).

Controls / affects: All authorization is expressed in terms of SignerId membership in policy roles. Also used for nonce replay protection.

RocksDB storage: Used inside keys via canon(signer).

```
// SCALE variant: tag:u8 + body
struct SignerId_Ed25519 { array<uint8_t,32> pubkey32; }; // tag 0
struct SignerId_Secp256k1 { array<uint8_t,33> pubkey33; }; // tag 1 (optional)
struct SignerId_Named { Hash32 id; }; // tag 2 (optional)

using SignerId = variant<SignerId_Ed25519, SignerId_Secp256k1, SignerId_Named>;

// Canonical signer bytes used for sorting + keys:
// canon(SignerId) = [tag_byte] || key_bytes
```

Signature

Purpose / why it exists: Binds TxEnvelope to SignerId for non-repudiation.

Controls / affects: Every custody command is authenticated; prevents unauthorized state transitions.

```
struct Signature_Ed25519 { array<uint8_t,64> sig64; }; // tag 0
struct Signature_Secp256k1 { array<uint8_t,65> sig65; }; // tag 1 (optional)
using Signature = variant<Signature_Ed25519, Signature_Secp256k1>;
```

TxEnvelopeV1

Purpose / why it exists: Standardizes signing, replay protection, and chain-domain separation.

Controls / affects: All state transitions (workspace/policy/intent/attestation) occur only via signed envelopes.

RocksDB storage: Nonce stored as: N| -> u64

```
struct TxEnvelopeV1 {
  uint16_t version;          // must be 1
  Hash32 chain_id;
  uint64_t nonce;            // replay protection per signer
  SignerId signer;
  TxPayloadV1 payload;
  Signature signature;
};

// Signing rule:
// signature = Sign( HASH("CUSTODY_TX_V1" || SCALE_ENCODE(envelope_without_signature)) )
```

Nonce record (RocksDB)

Purpose / why it exists: replay protection and deterministic ordering per signer.

Controls / affects: prevents duplicated or out-of-order custody commands from the same signing authority.

```
// Key
key_nonce(signer):
  return KeyBuilder().lit("N|").canon_signer(signer).finish()

// Value: u64 last_nonce
```

3) Governance boundary types

WorkspaceStateV1

Purpose / why it exists: Defines the top-level administrative domain (institution/tenant).

Controls / affects: Controls who can manage policies/roles and other high-privilege actions for that workspace.

RocksDB storage: Key: W| -> WorkspaceStateV1
key_workspace(ws)= "W|"+ws

```
struct WorkspaceStateV1 {
  Hash32 workspace_id;
  vector<SignerId> admin_set;     // sorted by canon(signer)
  uint32_t admin_quorum;          // 1..=len(admin_set)
  optional<Hash32> metadata_ref;  // off-chain pointer/hash
};

// Created/updated by CreateWorkspaceV1 (tx command)
struct CreateWorkspaceV1 { same fields as WorkspaceStateV1; };
```

VaultStateV1

Purpose / why it exists: Defines a custody boundary for assets (segregated/omnibus).

Controls / affects: Policies can scope rules to a vault; intents execute against a vault context.

RocksDB storage: Key: V|| -> VaultStateV1
key_vault(ws,v)= "V|"+ws+"|"+v

```
enum class VaultModelV1 : uint8_t { Segregated=0, Omnibus=1 };

struct VaultStateV1 {
  Hash32 workspace_id;
  Hash32 vault_id;
  VaultModelV1 model;
  optional<Bytes> label;
};

struct CreateVaultV1 { same fields as VaultStateV1; };
```

DestinationStateV1

Purpose / why it exists: Represents whitelisted transfer targets.

Controls / affects: Controls whether transfers can be executed to a given target (policy may require whitelisting).

RocksDB storage: Key: D|| -> DestinationStateV1
key_destination(ws,d)= "D|"+ws+"|"+d

```
enum class DestinationTypeV1 : uint8_t { Address=0, Contract=1 };

enum class ChainTag : uint8_t { Ethereum=0, Bitcoin=1, Solana=2, Other=255 };
```

```
struct ChainRefV1 {
  ChainTag tag;
  optional<Bytes> other;           // present if tag==Other
};

struct DestinationStateV1 {
  Hash32 workspace_id;
  Hash32 dest_id;
  DestinationTypeV1 dest_type;
  ChainRefV1 chain;
  Bytes address_or_contract;       // canonical per chain (you define)
  bool enabled;
  optional<Bytes> label;
};

struct UpsertDestinationV1 { same fields as DestinationStateV1; };
```

4) Asset registry

AssetStateV1

Purpose / why it exists: Normalizes asset identity and decimals across chains to make limits/audit deterministic.

Controls / affects: Controls whether an asset can be used in intents/policies; used during propose/execute validation and UI interpretation.

RocksDB storage: Key: AS| -> AssetStateV1
key_asset(id)= "AS|"+id

```
enum class AssetKindV1 : uint8_t {
  Native=0, ERC20=1, ERC721=2, ERC1155=3, Other=255
};

// Canonical 'where is this asset on its home chain'
struct AssetRef_NativeSymbol { Bytes symbol; };          // tag 0 (PoC-friendly)
struct AssetRef_ContractAddress { Bytes addr; };         // tag 1 (ETH: 20 bytes)
struct AssetRef_Composite { vector<Bytes> parts; };      // tag 2 (optional)
using AssetRefV1 = variant<AssetRef_NativeSymbol, AssetRef_ContractAddress, AssetRef_Composite>;

struct AssetStateV1 {
  Hash32 asset_id;
  ChainRefV1 chain;               // home chain
  AssetKindV1 kind;
  AssetRefV1 ref;                 // canonical identifier on chain
  optional<Bytes> symbol;          // "USDC"
  optional<Bytes> name;            // "USD Coin"
  uint8_t decimals;               // critical for interpretation
  bool enabled;
};

struct UpsertAssetV1 { same fields as AssetStateV1; };
struct DisableAssetV1 { Hash32 asset_id; }; // optional convenience
```

5) Policy model (roles, approvals, limits, timelocks, attestations)

PolicyScopeV1

Purpose / why it exists: Defines where a policy applies (workspace-wide or a specific vault).

Controls / affects: Controls which rules govern a given intent (based on its workspace/vault).

RocksDB storage: Used as a component of policy keys: PA|, PS||...

```
// tag 0: Workspace scope, tag 1: Vault scope
struct PolicyScope_Workspace { Hash32 workspace_id; };
struct PolicyScope_Vault { Hash32 workspace_id; Hash32 vault_id; };
using PolicyScopeV1 = variant<PolicyScope_Workspace, PolicyScope_Vault>;

// Deterministic scope key bytes used in RocksDB keys:
scope_key(scope):
  if Workspace(ws): "WS|" + ws
  if Vault(ws,v):   "VA|" + ws + "|" + v
```

RoleIdV1

Purpose / why it exists: Names authorization buckets for signers.

Controls / affects: Controls who is allowed to propose/approve/execute/administer/attest under a policy set.

```
enum class RoleIdV1 : uint8_t {
  Initiator=0, Approver=1, Executor=2, Admin=3, Auditor=4, Guardian=5, Attester=6
};
```

ApprovalRuleV1

Purpose / why it exists: Encodes multi-party governance for intents (M-of-N approvals + separation of duties).

Controls / affects: Controls when an intent becomes EXECUTABLE; blocks execution until satisfied.

```
struct ApprovalRuleV1 {
  RoleIdV1 approver_role;                 // usually Approver
  uint32_t threshold;                     // N-of-M
  bool require_distinct_from_initiator;
  bool require_distinct_from_executor;
};
```

LimitRuleV1

Purpose / why it exists: Caps risk exposure per transaction by asset.

Controls / affects: Controls execute-time (and optionally propose-time) validation for Transfer amounts.

```
struct LimitRuleV1 {
  Hash32 asset_id;
  Amount per_tx_max;
};
// policy.limits must be sorted by asset_id bytes
```

OperationTypeV1

Purpose / why it exists: Classifies the kind of custody action so policies can be action-specific.

Controls / affects: Controls which policy rule applies to an intent's action.

```
enum class OperationTypeV1 : uint8_t { Transfer=0, ContractCall=1, RawSign=2 };
```

TimelockRuleV1

Purpose / why it exists: Imposes a delay window for oversight and cancellation before execution.

Controls / affects: Controls IntentState.not_before_ms and blocks ExecuteIntent until satisfied.

```
struct TimelockRuleV1 {
  OperationTypeV1 operation;
  DurationMs delay_ms;
};
```

DestinationRuleV1

Purpose / why it exists: Allows policy to require destination whitelist enforcement.

Controls / affects: Controls whether ExecuteIntent must validate DestinationState enabled/existing.

```
struct DestinationRuleV1 {
  bool require_whitelisted;
};
```

ClaimTypeV1

Purpose / why it exists: Names compliance claims the system can require for execution.

Controls / affects: Controls which attestations must exist for an intent to execute.

```
// SCALE variant
struct ClaimType_KyBVerified {};   // tag 0
struct ClaimType_SanctionsClear {}; // tag 1
struct ClaimType_TravelRuleOk {};   // tag 2
struct ClaimType_RiskApproved {};   // tag 3
struct ClaimType_Custom { Hash32 id; }; // tag 255
using ClaimTypeV1 = variant<ClaimType_KyBVerified, ClaimType_SanctionsClear,
                            ClaimType_TravelRuleOk, ClaimType_RiskApproved, ClaimType_Custom>;
```

ClaimRequirementV1

Purpose / why it exists: Defines the compliance evidence needed to permit execution.

Controls / affects: Controls ExecuteIntent gating: each required claim must be satisfied by an active attestation.

```
struct ClaimRequirementV1 {
  ClaimTypeV1 claim_type;
  optional<TimestampMs> min_valid_until_ms;
  optional<vector<SignerId>> trusted_issuers; // if present, sorted canon(signer)
};
// required_claims sorted by canonical encoding of claim_type
```

PolicyRuleV1

Purpose / why it exists: Bundles all controls for a given operation type.

Controls / affects: Controls the entire propose/approve/execute flow for intents of that operation.

```
struct PolicyRuleV1 {
  OperationTypeV1 operation;
  ApprovalRuleV1 approvals;
  vector<LimitRuleV1> limits;                 // sorted by asset_id
  optional<TimelockRuleV1> timelock;
  DestinationRuleV1 destination;
  vector<ClaimRequirementV1> required_claims; // sorted
};
// rules sorted by operation
```

PolicySetStateV1 + ActivePolicyPointerV1

Purpose / why it exists: Defines the full governance/compliance regime, versioned for audit stability.

Controls / affects: Controls who can do what, what approvals/limits/timelocks are required, and what attestations must exist.

RocksDB storage: PS||| -> PolicySetStateV1
PA| -> ActivePolicyPointerV1
key_policy_set(scope,id,ver)= "PS|"+scope_key+"|"+id+"|"+u32_le(ver)
key_policy_active(scope)= "PA|"+scope_key

```
struct PolicySetStateV1 {
  Hash32 policy_set_id;
  PolicyScopeV1 scope;
  uint32_t version;
  vector<pair<RoleIdV1, vector<SignerId>>> roles; // sort by RoleId; members sorted
  vector<PolicyRuleV1> rules;                    // sorted by operation
};

struct ActivePolicyPointerV1 {
  Hash32 policy_set_id;
  uint32_t version;
};

// Commands:
struct CreatePolicySetV1 { same fields as PolicySetStateV1; };
struct ActivatePolicySetV1 { PolicyScopeV1 scope; Hash32 policy_set_id; uint32_t version; };
```

6) Custody intent lifecycle types

TransferParamsV1 + IntentActionV1

Purpose / why it exists: Represents the requested custody operation and parameters.

Controls / affects: Controls which PolicyRule applies (by operation) and drives validation for limits/destination/asset.

```
struct TransferParamsV1 {
  Hash32 asset_id;
  Amount amount;
  Hash32 destination_id;
};

struct IntentAction_Transfer { TransferParamsV1 p; }; // tag 0
using IntentActionV1 = variant<IntentAction_Transfer /*, ... future actions ... */>;
```

IntentStatusV1

Purpose / why it exists: Encodes the lifecycle stage of a custody request.

Controls / affects: Controls allowed next transitions (approve/execute/cancel) and prevents replays/duplicate execution.

```
enum class IntentStatusV1 : uint8_t {
  Proposed=0, PendingApprovals=1, Executable=2, Executed=3, Cancelled=4, Expired=5
};
```

IntentStateV1

Purpose / why it exists: Single authoritative record for the requested custody action + all gating conditions.

Controls / affects: Controls execution eligibility: status, timelock, expiry, approvals threshold, and required claims snapshot.

RocksDB storage: I||| -> IntentStateV1
key_intent(ws,vault,intent)= "I|"+ws+"|"+vault+"|"+intent

```
struct IntentStateV1 {
  Hash32 workspace_id;
  Hash32 vault_id;
  Hash32 intent_id;

  SignerId created_by;
  TimestampMs created_at_ms;

  TimestampMs not_before_ms;
  optional<TimestampMs> expires_at_ms;

  IntentActionV1 action;
  IntentStatusV1 status;

  // policy snapshot at propose-time
  Hash32 policy_set_id;
  uint32_t policy_version;

  uint32_t required_threshold;
  uint32_t approvals_count;

  // attestation requirement snapshot at propose-time
  vector<ClaimRequirementV1> required_claims; // sorted
};
```

ApprovalStateV1

Purpose / why it exists: Durable per-approver evidence record (auditable and deduplicated).

Controls / affects: Controls approvals_count (directly or via prefix-scan) and provides forensic audit of who approved.

RocksDB storage: IA|| -> ApprovalStateV1
key_intent_approval(intent,approver)= "IA|"+intent+"|"+canon(approver)

```
struct ApprovalStateV1 {
  Hash32 intent_id;
  SignerId approver;
  TimestampMs approved_at_ms;
};
```

Intent command transactions

Purpose / why it exists: Command surface area for moving intents through the lifecycle.

Controls / affects: Propose creates IntentState; Approve creates ApprovalState; Execute validates all gates then marks executed; Cancel terminates.

```
struct ProposeIntentV1 {
  Hash32 workspace_id;
  Hash32 vault_id;
  Hash32 intent_id;
  IntentActionV1 action;
  optional<TimestampMs> expires_at_ms;
};

struct ApproveIntentV1 { Hash32 workspace_id; Hash32 vault_id; Hash32 intent_id; };
struct ExecuteIntentV1 { Hash32 workspace_id; Hash32 vault_id; Hash32 intent_id; };
struct CancelIntentV1  { Hash32 workspace_id; Hash32 vault_id; Hash32 intent_id; };
```

7) Compliance evidence (attestations)

AttestationRecordV1

Purpose / why it exists: On-chain proof that off-chain compliance checks were performed by a trusted issuer.

Controls / affects: Controls ExecuteIntent gating when PolicyRule.required_claims is non-empty.

RocksDB storage: AT|||| -> AttestationRecordV1
key_attestation(ws,subject,claim,issuer)= "AT|"+ws+"|"+subject+"|"+SCALE(claim)+"|"+canon(issuer)

```
enum class AttestationStatusV1 : uint8_t { Active=0, Revoked=1 };

struct AttestationRecordV1 {
  Hash32 workspace_id;
  Hash32 subject;                 // recommended: vault_id for PoC
  ClaimTypeV1 claim_type;
  SignerId issuer;
  TimestampMs issued_at_ms;
  TimestampMs expires_at_ms;
  AttestationStatusV1 status;
  optional<Hash32> reference_hash; // off-chain case/doc hash
};
```

Attestation command transactions

Purpose / why it exists: Creates/updates/revokes compliance evidence without changing policy structure.

Controls / affects: Used by Attester/Admin roles to satisfy or withdraw claims required for intent execution.

```
struct UpsertAttestationV1 {
  Hash32 workspace_id;
  Hash32 subject;
  ClaimTypeV1 claim_type;
  TimestampMs expires_at_ms;
  optional<Hash32> reference_hash;
};

struct RevokeAttestationV1 {
  Hash32 workspace_id;
  Hash32 subject;
  ClaimTypeV1 claim_type;
  SignerId issuer;                // usually equals TxEnvelope.signer
};
```

8) Top-level payload variants (TxPayloadV1)

TxPayload is the command surface; each variant either creates/updates stored state or advances workflow.

```
// SCALE variant: tag:u8 + body
// IMPORTANT: never reorder; append new tags only.
using TxPayloadV1 = variant<
  CreateWorkspaceV1,          // tag 0
  CreateVaultV1,              // tag 1
  UpsertDestinationV1,        // tag 2
  CreatePolicySetV1,          // tag 3
  ActivatePolicySetV1,        // tag 4
  ProposeIntentV1,            // tag 5
  ApproveIntentV1,            // tag 6
  ExecuteIntentV1,            // tag 7
  CancelIntentV1,             // tag 8
  UpsertAttestationV1,        // tag 9
  RevokeAttestationV1,        // tag 10
  UpsertAssetV1,              // tag 11
  DisableAssetV1              // tag 12 (optional)
>;
```

9) Workflow: custody intent state machine and what types participate

```
                       ProposeIntentV1
                            |
                            | writes: I|ws|vault|intent -> IntentStateV1
                            | reads:  PA|scope, PS|..., AS|asset, D|dest (optional)
                            v
                  +---------------------+
                  | PROPOSED / PENDING  |
                  | approvals < thresh  |
                  +----------+----------+
                             |
                             | ApproveIntentV1 (repeat)
                             | writes: IA|intent|approver -> ApprovalStateV1
                             | updates: IntentStateV1.approvals_count (optional cache)
                             v  (when approvals >= threshold)
                        +----------+
                        |EXECUTABLE|
                        +----+-----+
                             |
                             | ExecuteIntentV1 checks:
                             |  - policy snapshot (IntentStateV1.policy_set_id/version)
                             |  - approvals >= required_threshold (ApprovalStateV1 records)
                             |  - ctx.time >= not_before_ms (TimelockRuleV1)
                             |  - asset enabled (AS|asset_id -> AssetStateV1)
                             |  - per-tx limits (LimitRuleV1)
                             |  - destination allowed if required (D|... -> DestinationStateV1)
                             |  - attestations satisfied (AT|... -> AttestationRecordV1)
                             | writes: updates IntentStateV1.status = EXECUTED
                             v
                         +--------+
                         |EXECUTED|
                         +--------+

CancelIntentV1:
  PROPOSED/PENDING or EXECUTABLE -> CANCELLED (updates IntentStateV1)

Expiry (lazy evaluation on interaction):
  if now > expires_at_ms: -> EXPIRED (updates IntentStateV1)
```

What each stored type controls in the workflow:

- WorkspaceStateV1 / VaultStateV1: define governance boundaries for policies and intents
- PolicySetStateV1 / ActivePolicyPointerV1: define which rules apply; snapshot prevents history rewrite
- DestinationStateV1: allowlist gate for transfers (if policy requires)
- AssetStateV1: asset validity + decimals normalization; gates asset usage
- IntentStateV1: the canonical workflow object; holds all gates and state
- ApprovalStateV1: auditable evidence that approvals occurred; drives threshold satisfaction
- AttestationRecordV1: auditable compliance evidence; gates execution
- Nonce record: replay protection for *all* commands