Custody Protocol — Event Subscription (Push Model) Roadmap

Goal: Provide a developer-friendly push model where clients can subscribe to custody events and receive updates in real time.

Transport assumption: CometBFT RPC WebSocket subscriptions to indexed ABCI events. This is a node-local push feed (clients connect to one or more RPC nodes).

1) What CometBFT Provides (High-level)

CometBFT supports subscribing to events over WebSocket via the `subscribe` RPC method with an event query.
Applications can attach events (type + attributes) to transaction results and to blocks; Comet can index these events and allow clients to query/subscribe based on them.

```
Example WebSocket subscription (JSON-RPC):
{
  "jsonrpc": "2.0",
  "method": "subscribe",
  "id": 1,
  "params": { "query": "tm.event='Tx'" }
}
```

2) Design Principles for Custody Events

1. Auditor-friendly: events must correspond to immutable on-chain artifacts (IntentState, ApprovalState, AttestationRecord).
2. Deterministic: event attribute values must be derived solely from transaction input + pre-state + post-state (no wall clock strings, no randomness).
3. Minimal: emit a small number of well-defined events with stable attribute keys.
4. Queryable: choose attributes that developers naturally filter on (workspace_id, vault_id, intent_id, asset_id, destination_id, status).

3) Event Taxonomy (V1)

Event names are strings. Use the prefix `custody.` and use one event per major workflow transition.

Event Type	Emitted When	Controls / Why it exists
custody.workspace.created	CreateWorkspace committed	Notify integrators and auditors that a governance domain exis
custody.vault.created	CreateVault committed	Notify that a custody boundary (vault) is created
custody.policy.activated	ActivatePolicySet committed	Notify that enforcement rules changed for a scope
custody.asset.upserted	UpsertAsset committed	Notify asset registry change (decimals/enablement)
custody.destination.upserted	UpsertDestination committed	Notify allowlist changes impacting execution

custody.intent.proposed	ProposeIntent committed	Signals new custody request; primary feed event
custody.intent.approved	ApproveIntent committed	Signals approval evidence added
custody.intent.executable	ApproveIntent results in threshold met	Signals that timelock + other gates are now the remaining blo
custody.intent.executed	ExecuteIntent committed	Signals final authorization/execution state reached
custody.intent.cancelled	CancelIntent committed	Signals request terminated
custody.attestation.upserted	UpsertAttestation committed	Signals compliance evidence added/updated
custody.attestation.revoked	RevokeAttestation committed	Signals compliance evidence revoked

4) Standard Event Attributes (V1)

All custody events should include a consistent set of attributes so developers can filter subscriptions without memorizing per-event schemas.

```
Common attributes (strings; values are hex or decimal strings):
custody.workspace_id     = <hash32 hex>
custody.vault_id         = <hash32 hex>             (if applicable)
custody.intent_id        = <hash32 hex>             (if applicable)
custody.asset_id         = <hash32 hex>             (if applicable)
custody.destination_id   = <hash32 hex>             (if applicable)
custody.policy_set_id    = <hash32 hex>             (if applicable)
custody.policy_version   = <u32 decimal string>     (if applicable)
custody.status           = proposed|pending|executable|executed|cancelled|expired
custody.signer_scheme    = ed25519|secp256k1|named
custody.signer_id        = <hex (pubkey bytes or id)>
custody.height           = <block height decimal>   (optional convenience)
custody.tx_hash          = <hex>                    (optional convenience)

Recommended convenience attribute:
custody.event            = <event type string>      (duplicate of Event.type for simpler queries)
```

5) Mapping: Which Types Drive Which Events

```
Examples:
- custody.intent.proposed:
    emitted after writing IntentStateV1 (I|ws|vault|intent)
- custody.intent.approved:
    emitted after writing ApprovalStateV1 (IA|intent|approver)
    and updating IntentStateV1.approvals_count
- custody.intent.executable:
    emitted when IntentStateV1.status transitions to EXECUTABLE
- custody.attestation.upserted:
    emitted after writing AttestationRecordV1 (AT|...)
```

6) Example Subscriptions (Developer UX)

Developers subscribe via WebSocket to an RPC node. Queries filter by event attributes.

```
Subscribe to all executed intents in a vault:
tm.event='Tx' AND custody.vault_id='<vault_hex>' AND custody.status='executed'

Subscribe to all events for a specific intent:
tm.event='Tx' AND custody.intent_id='<intent_hex>'
```

```
Subscribe to all policy activations for a workspace:
tm.event='Tx' AND custody.workspace_id='<ws_hex>' AND custody.event='custody.policy.activated'
```

7) Roadmap Plan (Phased)

Phase A (later, low lift):
1) Add event emission to deliver_tx handlers (ABCI FinalizeBlock tx_results events).
2) Enable transaction indexing on RPC nodes for the custody.* attributes you care about.
3) Publish a short "How to subscribe" guide with example queries.

Phase B (DX improvements):
4) Publish an SDK helper to build subscription queries and decode events.
5) Add an optional lightweight gateway that maintains WebSocket connections and exposes:
• SSE streams
• Webhooks fan-out
• Replay-from-height (best-effort) using tx search

Phase C (enterprise reliability):
6) Durable eventing with an external message bus (Kafka/NATS) fed by a gateway.
7) Backfill and near-exactly-once semantics (outside CometBFT core).

8) Caveats / Non-goals

• CometBFT push is not a guaranteed-delivery messaging system. Clients must reconnect and may miss events if disconnected.
• Indexing controls queryability; avoid indexing overly high-cardinality data.
• Event attributes are not consensus-critical; treat them as UX and indexing metadata.