Haley Life Cycle

Purpose

This document explains the Haley.Lifecycle embedded workflow concept so it can be reused as a stable reference across design iterations (database schema, engine APIs, monitoring, idempotency, and micro work-items).

High-Level Workflow Overview

1. Actors

Application

- Owns business entities (for example: *RegistrationSubmission, PrequalificationSubmission*).
- Owns business logic, side-effects, and any human-driven processes (approvals, reviews).
- Does not need to know workflow states or policy internals.

LCEngine (Embedded Library)

- Owns macro workflow truth: instances, current state, transitions, transition logs, timeouts.
- Imports workflow definition JSON into the database.
- Loads policy JSON into an in-memory cache (per definition version) to decide which application actions to invoke. (Policies are optional)

LCMonitor (Embedded Library)

- Runs on a time-based loop.
- Handles state timeouts (stored in the lifecycle database).
- Handles macro acknowledgement monitoring (*delivery awareness / crash resume*) by asking LCEngine to re-notify or re-trigger.

2. Core Contracts

Engine Events (Macro)

- Numeric eventCode values that drive state transitions.
- Raised by application, monitor, or engine itself.

Macro trigger call:

```
Trigger (instanceId, eventCode, requestId, metadata)
```

App Events (Actions)

- String codes such as APP.* mapped to application methods.

- Application methods are marked using:

  `LC_APP_EVENT("APP.XYZ")`

Macro Acknowledgement

- Confirms the application is aware that a macro transition occurred.
- One macro transition maps to one macro acknowledgement.
- Macro acknowledgement is delivery awareness; it is not micro work completion.

Micro Snapshots (Mini-State Items)

- Dumb model-of-record stored by the engine.
- Stored in an engine table like `instance_state_items.`
- Identity uses:
  - `item_key` (string)
  - optional `item_hash` (hash of `payload_json` for dedupe)
- Engine stores the data; the application writes/updates it; the engine does not interpret it.

3. Startup Sequence (Initializer)

1. Application calls `LCEngineInitializer.Initialize().`
2. Initializer:
   - Reads database settings from `appsettings.json`
   - Ensures database/schema exists
   - Imports workflow definitions from `./lifecycle_config/` into lcstate tables
   - Loads policy JSON into LCEngine memory cache (definition_name + definition_version)
   - Scans assemblies for methods that has the attributes `LC_APP_EVENT` and registers them via an `IAppActionInvoker`

4. Runtime Sequence (Instance → Macro Transitions → App Actions)

1. A submission occurs in the application.

2. Application creates a lifecycle instance:
   `CreateInstance(defName, defVersion, externalRef=submissionRef, metadata)` → instanceId
3. Engine performs AutoStart internally via Trigger(...).
4. On every macro event:
   - `Trigger(instanceId, eventCode, requestId, metadata)`
   - Engine validates and persists state and transition log.
5. After a transition:
   - If a policy route exists, engine invokes one or more app actions (short-running).
   - If no policy route exists, engine emits a generic `TransitionOccurred` notice.
6. Application acknowledges macro transition awareness (macro ack).
7. Application owns micro progression:
   - App updates micro snapshots (`instance_state_items`) via engine persistence methods.
   - App evaluates quorum/conditions and triggers the next macro engine event when ready.
8. LCMonitor runs:
   - Triggers timeout events via engine (idempotent)
   - Re-notifies on missing macro acknowledgements

Detailed Discussions & Future Enhancements

1. Definitions vs Policies

Definition

- Definitions (states, events, transitions, timeouts) are stable and persisted.
- Policies are loaded into memory and used to decide which application actions to invoke.
- Policy contents are treated as opaque inputs for application logic.

Example

- Definition: UnderEvaluation exists and has timeout settings in DB.
- Policy route: UnderEvaluation on enter invokes APP.EVAL.SETUP.

Question
Why keep policies in memory?

Answer
Policies change frequently and can be environment/app specific. Keeping them in memory avoids definition churn and keeps behavior flexible.

Future enhancement
Persist policy fingerprint (hash/guid) with transition logs so the system can reconstruct which rules were active historically.

---

2. Instance Creation and AutoStart

Definition

- Application creates an engine instance and receives instanceId.
- Engine moves from initial state to the next state automatically.

Example

- App: new submission created
- Engine: instance created in Initial, then AutoStart transitions to WorkflowSelection

Question
Should AutoStart be delayed or immediate?

Answer
Prefer internal AutoStart trigger (immediate or queued) with idempotency. Time delay introduces race conditions.

Future enhancement
Use an internal engine queue so AutoStart and internal tasks are consistent and retryable.

3. Macro Transition and Idempotency (requestId)

Definition

- Macro transitions are triggered via Trigger(instanceId, eventCode, requestId, metadata).
- Idempotency is enforced by storing requestId and preventing duplicate processing.

Example

- App triggers EvaluationCompleted but times out and retries.
- Reusing the same requestId prevents duplicate transitions.

Question
Why do we need requestId if transition log already has a unique ID?

Answer
Transition log ID is generated after processing. requestId lets the engine detect “already processed” across retries.

Question
Who generates requestId?

Answer

- Application generates one per major business action and reuses it on retry.
- Monitor generates deterministic requestId for timeout ticks.
- Engine internal follow-ups can reuse stable identifiers.

Future enhancement
Return “already processed” with the original transition result payload.

4. Macro Acknowledgement (Awareness, Not Completion)

Definition

- Macro ack confirms application is aware the transition happened.
- It does not represent completion of micro actions.

Example

- Engine transitions to UnderEvaluation.
- Application receives notice and acknowledges.
- Even if emailing fails later, application was still aware of transition.

Question
Why track macro ack if embedded?

Answer
Crashes/restarts can cause missed notifications or repeated deliveries. Macro ack enables safe re-notify.

Future enhancement
Include last-delivery timestamp, retry count, and backoff.

5. App Action Invocation (App Events)

Definition

- Engine invokes app actions based on policy routing.
- App actions are registered from methods marked with LC_APP_EVENT.
- Engine calls via IAppActionInvoker.

Example

- WorkflowSelection on enter invokes APP.WF.SELECTION.REQUIRED.
- Method performs short setup work, then returns.

Question
Why not hardcode calls inside engine?

Answer
Invoker abstraction preserves separation of concerns and supports future service externalization.

Future enhancement
Support sequential vs parallel invocation controlled by policy routing configuration.

6. Micro Work Items Snapshot (instance_state_items)

Definition

- Engine-side mini-state snapshot for visibility/audit summary.
- Not a full audit trail.

Identity (Current Pattern)

- item_key: stable string (recommended)
- item_hash: optional hash of payload_json for dedupe

Example (UnderEvaluation with 3 engineers)

- *approver:201 Pending*
- *approver:202 Pending*
- *approver:203 Pending*

  Later:
- *approver:201 Approved*
- *approver:202 Approved*

  Workflow moves forward even if approver:203 remains Pending.

Question
Why not hash-only identity?

Answer
Items are mutable; stable item_key is needed for deterministic updates.

Future enhancement
Snapshot items at macro state exit into a history table.

7. Who Decides Quorum / Completion?

Current Pattern

- Engine is policy-opaque, so it cannot compute quorum.
- Application decides and triggers the macro engine event.

Example (Quorum 2 of 3)

- App updates audit + snapshot on each approval.
- When approvals reach 2, app triggers EvaluationCompleted.

Pattern A (Suggestion: Recommended)

- App checks completion immediately on every micro update and triggers macro event.

Pattern B (Suggestion)

- Engine emits StateWorkChanged notice whenever micro snapshot changes.
- App listens, recomputes completion, triggers macro event if complete.

Pattern C (Suggestion)

- App checks completion only at timeout.
- Simpler but can delay workflow progress.

8. LCMonitor (Timeouts + Crash Resume)

Definition

- Reads timeout config; triggers timeout events via engine.
- Re-notifies on missing macro acknowledgements.

Example

- UnderEvaluation timeout occurs → monitor triggers timeout event code.

Question
How to avoid repeated timeout triggers?

Answer
Monitor uses deterministic requestId per timeout tick; engine dedupes by (instance_id, requestId).

Future enhancement
Retry backoff and last-attempt markers.

9. Data Ownership Summary

Lifecycle DB (Engine-owned)

- Instances, current state
- Transition logs
- Definitions and timeouts

Engine Visibility Tables

- instance_state_items (micro snapshot)

Application DB (App-owned)

- Full audit trail (who approved, comments, reads)

- Operational work items and UI queues

10. Common Mistakes and Improvements

Mistake
Treating macro acknowledgement as micro action completion.

Fix
Macro ack = awareness; micro work is separate (snapshot + app audit).

Mistake
Expecting engine to move forward automatically while engine is policy-opaque.

Fix
Application evaluates micro rules and triggers macro event.

Improvement
Keep IAppActionInvoker abstraction to support later service extraction.

Appendix — Key Terminology

- **Macro**: lifecycle state machine level (states/transitions/events)
- **Micro**: application-driven internal sub-steps under a macro state
- **Macro Ack**: application awareness of a transition
- **Micro Snapshot**: instance_state_items current/final statuses of sub-items
- **requestId**: idempotency key for Trigger calls