Technical Report: The Calc-Engine Golden-Master Regression Guard

Prepared for: Office of the CIO
System: AI Retirement Income Planner (v7)
Subject: Design, coverage, and operational role of the golden-master self-test
Date: 2026-06-15
Classification: Internal — Engineering Assurance

1. Executive summary

The planner is a financial-projection product: every screen, report, and AI recommendation is downstream of one calculation engine (`calcAllPhases → calcPhase → simPhase`). A silent arithmetic regression in that engine would not throw an error or crash a page — it would quietly produce *plausible but wrong* retirement numbers, which is the most damaging failure mode a financial tool can have.

The **golden-master self-test** is the control that mitigates this risk. It re-runs the live calculation engine against **20 frozen scenarios** and compares **12 financial outputs per phase** against known-good ("golden") values captured from an approved build. Any deviation — even by a single dollar — fails the check and names the exact scenario, phase, and metric that drifted.

The control runs in two places: interactively in the browser (`?selftest=1`) and as an **automated headless gate** on every build. It is deterministic, read-only, and adds no runtime dependencies to the shipped product. In its current state, **all 20 scenarios pass**, establishing a verified baseline.

2. Why this control exists (the business risk)

Risk	Without the guard	With the guard
Silent calculation drift	A refactor or feature changes a tax/RMD/balance result; no visible symptom; ships to customers	Build fails; exact drifted figure surfaced before release
Regression in an edge regime (UK/Canada/Australia, NIIT, RMD)	Only caught if someone manually re-checks that exact path	Each regime is a permanently-guarded fixture
Eroded trust in a paid product	Customers make real retirement decisions on wrong output	Output is regression-locked to an audited baseline

The planner ships as a single self-contained HTML file with no server and no telemetry. There is no production monitoring layer to catch a bad number after the fact, which makes **pre-release verification the only line of defense**. The golden master is that line.

3. How it works (architecture)

3.1 The three inputs

1. **Fixtures** (`SELFTEST_FIXTURES` , `src/03-app.js`) — 20 named scenarios. Each is built by deep-merging an override patch (`ov`) over the canonical defaults (`D_USD`), optionally pinning a national tax regime (`cur: 'CAD' | 'AUD'`) and supplying a lump-sum schedule (`lumps`).
2. **Metrics** (`SELFTEST_KEYS`) — the 12 per-phase outputs that are compared (see §4).
3. **Golden values** (`SELFTEST_GOLDEN`) — the frozen expected output for every fixture × phase × metric, captured from an approved build.

3.2 Execution

`_selfTestMetrics()` runs each fixture through the **real** `calcAllPhases` (not a copy), with three determinism safeguards:

- **Pinned horizon:** every run uses a fixed end age (90), so results never depend on today's date.
- **Pinned currency/residence:** `activeCurrency` is set per fixture and **restored in a `finally` block**, so a test run can never leak state into the live application.
- **Isolated inputs:** each fixture plan is a `JSON.parse(JSON.stringify(D_USD))` deep copy — the test never mutates live user state (it is strictly read-only).

`runSelfTest()` then diffs actual vs. golden using **exact integer equality** (all figures rounded to whole units). On the first mismatch it stops and reports `Phase N, <metric>: <actual> vs golden <expected>`.

3.3 Two run modes

- **Interactive:** open any build with `?selftest=1` . An overlay reports pass/fail with a per-fixture table. Used during development and manual verification.
- **Automated gate:** `tools/run_selftest.py` loads the newest build in headless Chromium (Playwright), reads the machine-readable `window.__selfTestResult` , prints a per-fixture summary, and **exits non-zero on any drift**. It is wired into the build via `python python\build.py --selftest` , making it suitable for a CI pipeline. A temporary local HTTP server is used so the page loads under an `http://` origin consistent with the app's Content-Security-Policy.

4. Coverage

4.1 The 12 guarded metrics (per phase)

`net` (net monthly income), `tax` , `b401k` , `bcash` , `bEq` , `bRoth` , `bSuper` (ending balances by account), `health` (healthcare cost), `rmd` (required minimum distribution), `ftc` (UK foreign tax credit), `niit` (net investment income tax), `magi` (modified AGI).

This was deliberately widened beyond the original six (balances + net/tax) so the fixtures actually exercise the healthcare-cost, RMD, UK-treaty FTC, NIIT, and MAGI code paths rather than just ending balances.

4.2 The 20 scenarios

Group	Fixtures	Code paths guarded
Core tax regimes	USD default · GBP/UK resident · Foreign resident · MFJ	US single/joint brackets, UK treaty + FTC, expat (US-healthcare disabled)
National regimes (currency-gated)	CAD resident (CPP/OAS) · AUD resident (Super/Age Pension)	Canadian and Australian tax + benefit models
Social Security timing	Claimed at 70 · Claimed at FRA 67	Claim-age adjustment, phase-count change (6 vs 5 phases)
Age / horizon edges	Early retirement (55) · Replan at current age 62	Pre-59½ rules, mid-plan replan arithmetic
Balance edges	High 401k (RMD stress) · Portfolio depletion · High net worth	RMD scaling, depletion-to-zero, large-balance compounding
Income & conversions	Roth conversion ladder · Part-time income · US pension stream · NIIT (high MAGI+gains)	Roth conversions, earned income, pension streams, NIIT threshold
Lump sums	Inflow · Outflow	Per-phase lump-sum handling
Goals overlay	Goals enabled	Confirms core phase math is unchanged when the goals layer is on

This matches the surface area an external reviewer flagged as previously unguarded: four national tax regimes, SS claim-age splits, RMD/NIIT/ACA, early-retirement + replan, Roth conversions, part-time + pension income, lump sums, and the goals overlay.

---

5. Maintenance and governance

- **When the engine changes intentionally**, golden values must be re-baselined: open a build with `?selftest=1`, run `_selfTestMetrics()` in the console, and paste the result into `SELFTEST_GOLDEN`. This is a **deliberate, reviewable act** — the diff in version control shows exactly which figures moved and by how much, creating an audit trail of every intended change to financial output.
- **Ownership:** the fixtures, keys, and golden values live in `src/03-app.js`; the headless gate in `tools/run_selftest.py`. Both are version-controlled.
- **Cost:** zero runtime cost (the test code ships dormant, activated only by the `?selftest=1` query flag) and near-zero maintenance cost outside of intentional re-baselining.

---

6. Limitations (honest scope)

1. **Characterization test, not a correctness proof.** It locks output to a *previously approved* baseline. It guarantees *no unintended change*; it does not independently prove the baseline itself is tax-accurate. Correctness of the baseline rests on the separate domain validation behind those golden numbers.
2. **Integer-rounded comparison.** Sub-dollar drift is not detected. This is intentional (avoids floating-point noise) and immaterial at planning scale, but worth stating.
3. **Fixed set of scenarios.** Coverage is only as good as the 20 fixtures. New product capabilities require new fixtures or they ship unguarded.

4. **Headless gate requires Playwright/Chromium.** The automated mode needs a one-time `pip install` + browser download; the interactive `?selftest=1` mode has no such dependency.

---

7. Recommendations

1. **Make the headless gate a required CI step** (`python python\build.py --selftest`) so no build with drifted figures can be released.
2. **Add a fixture whenever a new calculation path ships** (e.g., a new country regime or account type) — treat "new math without a new golden fixture" as an incomplete change.
3. **Require a reviewer sign-off on any `SELFTEST_GOLDEN` diff**, since that diff *is* the record of every intended change to customer-facing financial output.
4. **Periodically re-validate the baseline** against authoritative tax tables when brackets/thresholds update for a new tax year, then re-baseline deliberately.

---

8. Conclusion

The golden-master self-test is a low-cost, high-leverage assurance control precisely targeted at this product's worst failure mode: silently wrong money. It is deterministic, read-only, dual-mode (interactive + automated), and currently green across all 20 scenarios. With the headless gate enforced in CI and fixture coverage kept in step with new features, it provides defensible, auditable protection that the numbers customers rely on do not change unless we intend them to.

---

9. License & Usage Terms

The AI Retirement Income Planner and all associated materials — including this document and the distributed application file — are **proprietary and licensed, not sold**.

- **Modification is not permitted.** Altering, editing, decompiling, reverse-engineering, or creating derivative works from the file is prohibited.
- **Redistribution, resale, sublicensing, or sharing of this file with any third party is strictly prohibited.**
- **All rights reserved.**