# **M5 — TypeScript Core Builders & Types (≈ 220 steps)**

**Goal:** give UC-4 a robust **off-chain planning layer**: canonical route encoding, serialization/deserialization, deterministic hashing, off-chain policy mirroring, bundle building, ABI typing, and helpers for quoting/simulation.  
 **Assumptions:** M4 contracts (executor, vault, adapter, registry, policies) are deployed and stable.

## **A. Branch, scaffolding & baselines**

1. Create branch feature/m5-core-ts.
2. Tag repo pre-m5-cut at last green main.
3. Add .env.example entries: SIMULATION\_RPC\_URL=, DEFAULT\_SLIPPAGE\_BPS=100.
4. Add packages/core-exec if not already; ensure package.json with "type":"module".
5. Add "strict":true and "exactOptionalPropertyTypes":true in its tsconfig.
6. Create directories: src/bundle/, src/route/, src/quote/, src/policy/, src/abi/.
7. Add pnpm script "build:core-exec": "tsc -p packages/core-exec/tsconfig.json".
8. Update CI: add job ts-core-exec-build before CLI/test jobs.

## **B. ABI types generation & exports**

1. Create tools/codegen/generate.ts:  
   * run forge build to refresh ABIs,
   * collect from contracts/out/\*\*.json,
   * typechain target=viem,
   * output to packages/core-exec/src/abi-types.
2. Add pnpm script "codegen": "tsx tools/codegen/generate.ts".
3. In packages/core-exec/src/abi/index.ts, export factories and types.
4. Ensure @org/types exports unified ABI type definitions.
5. CI step: after forge build, run pnpm codegen.
6. Confirm pnpm -w typecheck passes with generated ABIs.

## **C. Canonical TypeScript types**

1. Create BundleTypes.ts:

* StepParams { tokenIn, tokenOut, amountIn, minOut, recipient, deadline, venueId, data: Hex }.
* Bundle { steps: StepParams[], tokenIn, tokenOut, minTotalOut, recipient, deadline }.

1. Mirror contract structs (ensure field order matches Solidity).
2. Add discriminated union: VenueStep = { venue:'univ3'|'noop'|...; params:StepParams }.
3. Export PolicyConstraints { maxHops, slippageBps, deadlineSecs }.
4. Add Zod schemas for runtime validation.
5. Unit test: round-trip parse → stringify → parse stable.

## **D. Encoding & hashing**

1. Add RouteEncoder.ts.
2. Implement encodeStep(params) → 0x… ABI-encoded.
3. Implement decodeStep(bytes) → StepParams.
4. Add encodeBundle(bundle) → bytes.
5. Add hashBundle(bundle) → keccak256 of canonical encoding.
6. Ensure deterministic order of steps.
7. Add fixture tests: same bundle → same hash; permuted steps → different hash.
8. Write golden test vector in JSON (commit to repo).

## **E. Path building (Uniswap V3 & beyond)**

1. Add RouteBuilder.ts.
2. Implement buildUniv3Path(tokens[], fees[]) → Hex with ABI correctness.
3. Add helper deriveStep(tokenIn, tokenOut, amountIn, slippageBps, deadline, venue, data) → StepParams.
4. Implement buildSteps(routeSpec) where routeSpec = {tokens, fees, amountIn, slippage}.
5. Apply slippage guard: minOut = quotedOut\*(1-slippageBps/10000).
6. Add buildBundleFromSteps(steps[], tokenIn, tokenOut, recipient, deadline).
7. Unit test single-hop and multihop encodings.

## **F. Off-chain quoting & simulation glue**

1. Add quote/index.ts abstraction.
2. Define QuoteFn = (step, amountIn) => Promise<bigint>.
3. Implement quoteUniv3 via viem client & Quoter ABI.
4. Implement quoteNoOp returning amountIn passthrough.
5. Add dispatcher: quoteStep(step) → correct fn by venue.
6. Add simulateBundle(bundle):

* sequentially apply quoteStep,
* accumulate expectedOut,
* check minOut constraints off-chain.

1. Return simulation report: {expectedOut, perStepOut[], pass:boolean, errors[]}.

## **G. Policy mirroring (TS side)**

1. Add policy/index.ts.
2. Mirror PolicyConstraints with Zod validation.
3. Implement validateBundle(bundle, policy) off-chain:

* hops ≤ maxHops,
* slippage ≤ bound,
* deadline within bound.

1. Mirror Oracle drift check stub (wire real in M6).
2. Unit test: invalid bundles rejected early.

## **H. RPC client & providers**

1. Add src/providers.ts.
2. Export getPublicClient(rpcUrl) using viem.
3. Add support for SIMULATION\_RPC\_URL env fallback.
4. Add caching layer for clients keyed by chainId.
5. Add test: invalid URL → throws.

## **I. Utilities**

1. Add units.ts with parseUnits, formatUnits.
2. Add logger.ts with LOG\_LEVEL env-driven logging.
3. Add retry.ts (generic wrapper with backoff).
4. Add addresses.ts to load deployments JSON & env.
5. Unit test each util separately.

## **J. CLI integration hooks**

1. Update CLI commands to import from core-exec instead of duplicating.
2. Replace old route builder with new canonical functions.
3. Add CLI route simulate → calls simulateBundle & prints JSON.
4. Add CLI route hash → prints bundle hash.
5. Add CLI route inspect → decodes step bytes back to human-readable.
6. Unit test CLI commands with snapshots.

## **K. Tests — TypeScript**

1. Add packages/core-exec/test/encoder.spec.ts.
2. Test: encoding matches Solidity decode.
3. Add hash.spec.ts: deterministic results.
4. Add builder.spec.ts: step derivation, slippage calc.
5. Add simulate.spec.ts: quotes mocked; totals consistent.
6. Add policy.spec.ts: constraints enforced.
7. Add providers.spec.ts: rpc fallback.
8. Add units.spec.ts: parse/format correctness.
9. Coverage ≥ 85%.

## **L. Tests — Solidity cross-check**

1. Add contracts/test/RouteEncoder.t.sol.
2. Use TS built bytes[] → decode in Solidity → assert equality.
3. Hash computed in TS vs Solidity → match.
4. Add fuzz: random steps, compare encodings TS vs Solidity.

## **M. Documentation**

1. Add docs/dev/m5-overview.md with scope & diagrams.
2. Add docs/core/route-encoding.md: ABI layout, hash definition.
3. Update README.md with pnpm route simulate example.
4. Add examples/bundle.simple.json and examples/bundle.multihop.json.
5. Add runbook docs/runbooks/ts-sim.md.

## **N. CI & pipelines**

1. Update .github/workflows/ci.yml:

* add job ts-core-tests,
* run pnpm build:core-exec,
* run vitest with coverage,
* upload coverage report.

1. Add codecov integration optional.
2. Ensure golden vectors committed, diff them in CI.

## **O. Acceptance criteria**

1. Core-exec builds cleanly.
2. Encoding/decoding stable across TS & Solidity.
3. Deterministic hash matches Solidity.
4. Simulation runs with NoOp adapter and Uniswap V3 quoting.
5. Policy mirroring rejects invalid bundles off-chain.
6. CLI can simulate, hash, inspect a bundle end-to-end.
7. Docs provide clear examples.
8. CI passes green with coverage ≥ 85%.

## **P. Release prep**

1. Bump versions 0.5.0-m5 across core-exec and CLI.
2. Tag repo m5-core-ts.
3. Write release notes with summary: canonical encoding, hash determinism, simulation, CLI integration.
4. Merge once CI green + stakeholder demo passes.

### **Definition of Done for M5**

* ✅ Canonical TS types mirror Solidity structs
* ✅ Encoding/decoding works bidirectionally with contracts
* ✅ Deterministic bundle hash across TS & Solidity
* ✅ Off-chain policy mirroring & simulation stable
* ✅ CLI exposes simulate/hash/inspect commands
* ✅ Docs & examples ready for contributors
* ✅ CI green, release tagged