# **M6 — Adapter Client & CLI (≤ 260 steps)**

**Goal:** deliver a production-grade **Adapter Client** (UniswapV3-focused for MVP) and a **CLI** that can build, simulate, submit, and observe UC-4 bundles end-to-end, using the M5 core builders/types.  
 **Assumptions:** M0–M5 are green; contracts (executor, vault, registry/guards, Univ3 adapter) are deployed; ABIs/types are generated (M5); addresses.json has entries for the active network.

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

1. Create branch feature/m6-adapter-cli.
2. Tag repo pre-m6-cut at last green main.
3. In root .env.example, add:  
   * DEFAULT\_DEADLINE\_SECS=300, DEFAULT\_SLIPPAGE\_BPS=100, CONFIRMATIONS=1, CLI\_NETWORK=local, BUNDLE\_RECIPIENT=.
4. Ensure **packages** exist: packages/adapters-evm, packages/cli.
5. In packages/adapters-evm/package.json: "type":"module", exports set to ./dist/index.js.
6. In packages/cli/package.json: "type":"module", bin entry "uc4": "./dist/bin.js".
7. Add scripts:  
   * root: "build:adapters-evm", "build:cli", "dev:cli" (ts-node/tsx run).
8. Ensure tsconfig references M5 output (@core-exec/\*) paths compile cleanly.
9. Update CI to add jobs: adapters-evm-build, cli-build before tests.

## **B. Adapter Client: APIs and types**

1. Create packages/adapters-evm/src/UniswapV3AdapterClient.ts.
2. Import Bundle, StepParams, builders & encoders from @core-exec.
3. Define ClientConfig { chainId, executor, registry, vault, router, quoter, rpcUrl, confirmations }.
4. Add UniswapV3AdapterClient constructor (cfg: ClientConfig, provider = getPublicClient(cfg.rpcUrl)).
5. Define public methods (signatures only for now):  
   * getQuote(tokens: HexAddress[], fees: number[], amountIn: bigint): Promise<bigint>
   * buildSteps(tokens: HexAddress[], fees: number[], amountIn: bigint, slippageBps?: number, deadlineSecs?: number): Promise<StepParams[]>
   * buildBundle(args): Promise<Bundle>
   * submitBundle(bundle, signer): Promise<TxReceiptLike>
   * simulate(bundle): Promise<SimulationReport>
   * inspect(bundle): HumanReadableBundle
6. Add type HexAddress = \0x${string}`` and reuse Zod schemas from M5 for runtime guard.
7. Export a light AdapterVenueId = 'UNIV3' constant.

## **C. Adapter Client: quoting and path composition**

1. Implement getQuote() using M5 quoteUniv3 + Quoter address from cfg.
2. Validate inputs: tokens length = fees length + 1; fees ∈ {500, 3000, 10000} (or registry allowlist).
3. Implement buildSteps() via M5 buildUniv3Path and buildSteps helpers; clamp slippage with M5 clampSlippage.
4. Default slippageBps = env.DEFAULT\_SLIPPAGE\_BPS || policy.maxSlippageBps.
5. Default deadline = now + (env.DEFAULT\_DEADLINE\_SECS || 300).
6. Produce StepParams[] with minOut per hop using M5 simulation results.
7. Unit test: bad inputs revert with clear messages; good inputs produce path bytes of expected length.

## **D. Adapter Client: bundle build, inspect, and simulate**

1. Implement buildBundle() using M5 buildBundle, passing recipient (env or arg).
2. Compute minTotalOut from simulate(bundle) expectedOut \*(1 - slippageBps/10000).
3. Implement simulate(bundle) with M5 dispatcher; return { expectedOut, perStep, pass, errors }.
4. Implement inspect(bundle) → decodes steps using M5 decoders; returns { hops:[{tokenIn, tokenOut, fee, minOut}], sumMinOut, deadlineISO }.
5. Unit test: simulation with mocked quotes yields deterministic outputs; inspect prints human-readable.

## **E. Adapter Client: submission, receipts, and error surfacing**

1. Add submitBundle(bundle, signer) that:  
   * encodes calldata with M5 toCalldata (Bundle ABI),
   * sends tx to cfg.executor,
   * waits cfg.confirmations,
   * returns { hash, blockNumber, gasUsed, status }.
2. Catch contract reverts; parse custom errors (executor/adapter) to friendly CLI messages.
3. Retry policy: one safe retry on nonce too low or replacement underpriced with bump.
4. Unit test: mock provider to assert call data, to/from, gas limit presence, and receipt mapping.
5. Add estimateGas(bundle) using provider; expose --dry-run path in CLI (later section).

## **F. Address & policy resolution**

1. Add resolveConfig(network) that merges:  
   * config/addresses.json[network],
   * deployments/<network>.json,
   * .env overrides.
2. Implement fetchPolicy(venueId) view calls to registry (if ABI available) to pre-validate off-chain: maxHops, allowed fees, deadline bound.
3. If registry unavailable, proceed with env defaults but warn (non-blocking).
4. Unit test: missing addresses produce actionable errors; valid config returns typed config.

## **G. CLI: command skeleton**

1. Create packages/cli/src/bin.ts (or bin.js) that loads commander/yargs.
2. Add top-level commands: route build, route simulate, route exec, route inspect, route tail, route deposit, route balances, route policy.
3. Add global options: --network, --rpc, --confirmations, --recipient, --slippage-bps, --deadline-secs, --dry-run, --save.
4. Implement loadConfig() shared util to resolve network and addresses (Section F).
5. Wire graceful exit and non-zero code on failure; pretty errors.

## **H. CLI: route build**

1. Flags: --tokens 0xA,0xB,0xC, --fees 500,3000, --in 1.0, --decimals 18, --recipient, optional --slippage-bps, --deadline-secs.
2. Parse/validate inputs (Zod); convert human to wei using M5 parseUnits.
3. Instantiate UniswapV3AdapterClient with resolved config.
4. Call client.buildSteps() and client.buildBundle().
5. Print **bundle JSON** + **hex steps** + a summary table (tokens/fees/minOut/deadline).
6. If --save, write .cache/last-bundle.json.
7. Snapshot test: stable JSON structure and pretty output.

## **I. CLI: route simulate**

1. Accept --bundle <path> or rebuild from flags as in route build.
2. Call client.simulate(bundle); print { perStep, expectedOut, pass, reason? }.
3. Exit non-zero if pass=false unless --ignore-fail.
4. Snapshot test: deterministic formatting.

## **J. CLI: route exec**

1. Accept --bundle <path> or rebuild; require signer (private key or local account).
2. If --dry-run, only estimateGas and print calldata size + gas estimate.
3. Send with client.submitBundle; print hash, gasUsed, status, block.
4. If --confirmations N, await N; otherwise use cfg.confirmations.
5. Parse and print emitted BundleExecuted and per-hop AdapterSwapExecuted (M8 log decoders).
6. Exit non-zero on failure; surface revert reason using decoder mapping.
7. Integration test on local anvil: deposit → exec → verify balances.

## **K. CLI: route inspect**

1. Accept --bundle <path> or hex; uses client.inspect for readable hops.
2. Show per-hop {tokenIn, tokenOut, fee, minOut}, total min, deadline ISO.
3. Provide --json to print machine-readable object.
4. Snapshot test: formatting + JSON shape.

## **L. CLI: route tail**

1. Flags: --from-block, --to-block?, --follow, --max=1000.
2. Subscribe to BundleExecuted and AdapterSwapExecuted filtered by executor.
3. Print compact line per event; maintain .cache/uc4.history.json.
4. Handle reorgs: if --follow, re-check last K blocks on reorg events.
5. Test: mock provider events; verify decode and output lines.

## **M. CLI ops: route deposit, route balances, route policy**

1. **deposit**: flags --token 0x, --amount 1.0, --decimals; calls ERC20 approve→Vault deposit; prints vault balance.
2. **balances**: reads balances for --owner or default signer across allowlisted tokens.
3. **policy**: prints registry policy for venue UNIV3 (maxHops, slippage, deadline, fees).
4. Integration tests (local): mint mock tokens (dev script), deposit, and read balances.

## **N. Robust UX: defaults, scaling, and safety**

1. Autodetect decimals via ERC20 decimals() if --decimals not passed.
2. Default recipient to signer address if not provided; warn if empty.
3. If tokens array > 3 hops, refuse unless registry allows; print allowed max.
4. If deadline exceeds policy bound, shrink to bound + warn.
5. If slippage requested > policy bound, clamp + warn.
6. Print deterministic bundleHash (M5) after build to ease cross-system correlation.
7. Add --silent to suppress progress spinners for CI use.
8. Add --yes to skip confirmation prompts on large swaps.

## **O. Error handling & retry/resubmission**

1. Map common RPC/tx errors to friendly strings:

* insufficient funds, nonce too low, replacement underpriced, intrinsic gas too low.

1. On nonce too low, automatically refresh nonce and retry once.
2. On underpriced, bump priority fee by +15% once; cap by --max-priority-gwei.
3. Provide --no-retry flag to disable auto-retry.
4. Unit tests: simulated provider errors → expected retry behavior.

## **P. Config, networks, and environment switching**

1. Support --network local|fork|sepolia|mainnet → maps to addresses.json keys.
2. Allow --rpc to override network RPC.
3. If quoter/router missing for network, fail with actionable remediation (seed addresses or switch network).
4. Provide uc4 networks list (simple command) showing availability and critical addresses.
5. Snapshot test: networks output stable.

## **Q. Security & safety switches (MVP)**

1. Add --require-paused=false|true to enforce paused state before certain ops (owner ops only).
2. Protect **deposit**: refuse if executor not registered as approved spender in vault (reads view and warns).
3. Enforce **recipient** not zero, and not vault/executor addresses.
4. Add --max-amount-in guard to refuse large trades unless explicit override.
5. Document these switches in CLI help and README.

## **R. Tests — unit & integration (TypeScript)**

1. **Unit** (adapters-evm): quote/build/simulate/inspect/submit encode correctness with mocked provider.
2. **Unit** (cli): parse flags → action dispatch; JSON outputs snapshot stable.
3. **Integration** (local):  
   * boot anvil, deploy stack (scripts from earlier milestones),
   * mint mocks, deposit,
   * build 2-hop route, simulate, exec, tail logs,
   * assert balances and events.
4. Negative tests: invalid fee tier, exceeded hops, deadline expired, insufficient minOut → proper errors.
5. Coverage target: ≥ 85% for adapters-evm, ≥ 80% for cli (raise later).

## **S. CI & pipelines**

1. Extend .github/workflows/ci.yml:  
    - adapters-evm-unit (vitest),  
    - cli-unit,  
    - cli-integration (starts anvil service, runs scripted e2e).
2. Cache pnpm, foundry, and out/ artifacts keyed by lockfiles + foundry.toml.
3. Upload test reports and artifacts/m6-cli/\*.json (bundle examples, gas estimates).
4. Add pnpm -w build:adapters-evm && build:cli gates before tests.
5. Make cli-integration non-flaky: pin fork block or use local liquidity fixture.

## **T. Documentation**

1. docs/cli/overview.md: command matrix with examples.
2. docs/cli/recipes.md:  
   * single-hop exactInput,
   * two-hop multiswap,
   * dry-run + gas estimate,
   * deposit + execute.
3. docs/runbooks/ops-cli.md:  
   * pause/unpause,
   * freeze venue via registry (set maxHops=0),
   * health checks,
   * rollback (don’t exec; revoke spender).
4. Update README.md: quickstart with route build|simulate|exec|tail.
5. Add examples/ JSONs: bundle.single.json, bundle.multihop.json.
6. Link policy and address resolution behavior; include troubleshooting table.

## **U. Telemetry & observability hooks (CLI-side)**

1. Add optional --metrics-file reports/metrics/m6.cli.json to write:  
   * quotes, expected vs realized out, per-hop realized slippage (if on-chain readback), tx latency.
2. Add route report to summarize last N executions from .cache/uc4.history.json.
3. Document how to enable; keep off by default.

## **V. Performance & UX polish**

1. Cache ABI interfaces and decimals lookups across runs (.cache/abis.json, .cache/decimals.json).
2. Reduce RPC roundtrips by batching ERC20 calls (multicall if available).
3. Parallelize per-hop quotes in simulation where safe (respect path dependencies for amounts).
4. Progress spinners for long RPC operations; hide with --silent.
5. Pretty tables (width-safe) with truncation for addresses.

## **W. Safety validations at send time**

1. Before sending, re-quote last hop; if expected slippage > requested bound, prompt to continue unless --yes.
2. Enforce deadline > now by Δ ≥ 60s; otherwise extend or abort with hint.
3. Confirm network/chainId mismatch detection; refuse to send if provider chainId != cfg.chainId.
4. Unit tests cover these guards.

## **X. Backward/forward compatibility**

1. Read version() from executor (if implemented) and warn if incompatible with CLI version.
2. If bundle schema changes (M5 JSON schema), validate against the schema version in file and migrate fields if trivial.
3. Provide uc4 bundle upgrade --in old.json --out new.json for schema bump (no-op for MVP unless needed).
4. Tests: ensure older fixture files still pass or produce actionable migration hints.

## **Y. Acceptance criteria (M6)**

1. UniswapV3AdapterClient exposes: quote, buildSteps, buildBundle, simulate, inspect, submit; all unit-tested.
2. CLI supports: build, simulate, exec, inspect, tail, deposit, balances, policy.
3. End-to-end on local: deposit → build 2-hop → simulate pass → exec → tail events → balances increased.
4. Error surfaces are friendly (custom error decode) and actionable.
5. Coverage meets targets; CI fully green.
6. Docs include runnable examples and troubleshooting.
7. Metrics/report output optional but functional.

## **Z. Release prep**

1. Bump versions: adapters-evm and cli to 0.6.0-m6.
2. Tag repo m6-adapter-cli.
3. Changelog entry: features, commands, safety guards, examples.
4. Attach artifacts: example bundles, sample receipts, metrics json.
5. Merge once stakeholder demo passes.

### **Detailed build-out (141–220): concrete engineering actions**

1. Implement packages/adapters-evm/src/config.ts to load/validate ClientConfig from network/env.
2. Implement packages/adapters-evm/src/errors.ts mapping revert selectors → names → human messages.
3. Implement packages/adapters-evm/src/submit.ts housing estimateGas, submitTx, waitForReceipt.
4. Implement packages/adapters-evm/src/inspect.ts to decode steps & path bytes.
5. Implement packages/adapters-evm/src/policy.ts to fetch registry policy (view calls).
6. Export single entry index.ts re-exporting Client, types, errors, helpers.
7. In CLI, create src/cmds/build.ts using shared parse helpers.
8. Create src/cmds/simulate.ts, src/cmds/exec.ts, src/cmds/inspect.ts, src/cmds/tail.ts, src/cmds/deposit.ts, src/cmds/balances.ts, src/cmds/policy.ts.
9. Create src/cliConfig.ts (read .env, addresses.json, and flags).
10. Create src/format.ts (tables, address shortener, bigint format).
11. Create src/io.ts (read/write bundle JSON, schema validate).
12. Create src/logger.ts controlled by LOG\_LEVEL.
13. Create src/bin.ts with commander wiring; export default handler.
14. Unit tests (adapters-evm):  
     - client.quote.spec.ts (mock quoter call),  
     - client.build.spec.ts (path/steps),  
     - client.simulate.spec.ts,  
     - client.inspect.spec.ts,  
     - client.submit.spec.ts (mock provider).
15. Unit tests (cli):  
     - build.cmd.spec.ts, simulate.cmd.spec.ts, exec.cmd.spec.ts, inspect.cmd.spec.ts, tail.cmd.spec.ts.
16. Integration script packages/core-exec/src/run/localExecute.ts updated to invoke CLI internally (spawn) for smoke E2E.
17. Golden vectors: store examples/bundle.multihop.json + expected hash in packages/core-exec/test/\_\_vectors\_\_.
18. Add route hash command (thin wrapper of M5 hash) for quick comparison.
19. Add route dry-run alias for exec --dry-run.
20. Add route gas to only estimate and print gas/fees.
21. Implement decimals cache: src/cache/decimalsCache.ts keyed by token address+chain.
22. Implement ABI cache for events to speed decode.
23. Implement .cache/last-network.txt for default network fallback.
24. Add **health** subcommand (optional): uc4 health prints executor addr, paused state, registered UNIV3 adapter, policy summary.
25. Integrate paused check in exec: if paused, refuse with hint to unpause (operator flow).
26. Add --max-slippage-bps CLI guard; default to policy bound; refuse if over.
27. Add --min-total-out override; if provided, supersedes computed min.
28. Add --no-approve flag in deposit to skip ERC20 approval (advanced users).
29. Add ERC20 allowance check; if insufficient and --no-approve false, auto-approve exact amount.
30. Implement reorg handling in tail: maintain a min block pointer and reconcile on chainChanged/fork signals.
31. Implement --from-tx to tail from a specific tx hash (resolve block).
32. Add --out-json <path> to write results of build/sim/exec to a file for pipelines.
33. Add --stdin mode to accept bundle JSON piped in.
34. Add per-hop realized event check post-exec: parse AdapterSwapExecuted events and compute realized minOut satisfied; warn if close to limit.
35. Write minimal metrics (if enabled) to reports/metrics/m6.cli.json.
36. Add colorized output with fallback for non-TTY.
37. Ensure all commands support --json machine output; default is human.
38. Harden error mapping:  
    * decode custom errors using ABIs,
    * fallback to hex selector → name table,
    * final fallback prints raw data with hint to run inspect.
39. Map these to exit codes: 1 (validation), 2 (RPC/network), 3 (revert/policy), 4 (IO/FS).
40. CI: add cli-e2e-local job that runs:  
    * pnpm anvil:up,
    * deploy stack (scripts),
    * seed liquidity or load fork fixture,
    * uc4 route build --save,
    * uc4 route simulate .cache/last-bundle.json,
    * uc4 route exec .cache/last-bundle.json --yes,
    * uc4 route tail --from-block <deployBlock> --max 10.
41. Upload artifacts: last-bundle.json, receipt.json, tail.log.
42. Docs examples:  
    * **Example 1**: WETH→TOKEN single-hop,
    * **Example 2**: TOKEN→WETH→TOKEN multihop,
    * **Example 3**: dry-run + gas,
    * **Example 4**: deposit + balances.
43. Troubleshooting section: nonce/underpriced/gas errors; policy violations; missing addresses.
44. Final polish: ensure help texts list defaults (slippage, deadline, confirmations).
45. Add --version printing CLI + contract versions (if exposed) and chainId.
46. Add smoke script scripts/smoke/m6.sh chaining the typical workflow; CI uses it.
47. Ensure Windows/macOS paths work (avoid bash-only calls inside Node scripts).
48. Add “confirmed success” banner with a compact summary and next-steps link to docs.
49. On failure, print next diagnostic command suggestions (uc4 route inspect, uc4 route tail --from-tx ...).
50. Validate large integer safety: use bigint everywhere; never Numbers for amounts/wei.
51. Validate ESM only imports, no stray require.
52. Ensure CONFIRMATIONS respected in exec/wait; default to 1 on local, 2+ on testnets.
53. Add --timeout-ms to receipt wait; fallback to polling interval.
54. If simulation expectedOut is zero, block execution with explicit message (liquidity/price issue).
55. If tokens array contains duplicates implying a cycle, warn but allow if policy permits (for MVP keep simple: allow).
56. Add --policy-enforce-offchain=false|true default true; if false, allow build/exec even if off-chain validation fails (operators may override).
57. Print explicit **policy snapshot** used during build (hash, bounds) for traceability.
58. Implement uc4 route export --bundle <path> to flatten bundle + policy + addresses into a single portable JSON artifact for reproducible runs.
59. Implement uc4 route import --file <path> to run with that artifact ignoring local env (safe mode).
60. Wire SIGINT/SIGTERM handling to stop tails cleanly and save partial logs.
61. Ensure file writes are atomic (write temp + rename) to avoid corruption.
62. Add --gas-price-gwei and --max-fee-gwei for manual fee control; respect EIP-1559 where available.
63. Add fee sanity: if manual fees < network min, warn and bump unless --no-bump.
64. Add route whoami to print signer address and nonce (quick connectivity check).
65. Add route chain to print chainId, latest block, base/priority fee.
66. Create docs/cli/faq.md (10–12 real Q&As based on above errors).
67. Link docs from CLI help footer.
68. Ensure **no private keys** are ever logged; redact env/args if they accidentally include secrets.
69. Use minimal telemetry; off by default; clearly document opt-in.
70. Run full **local E2E** manually:  
    * pnpm anvil:up,
    * deploy stack,
    * mint & deposit,
    * uc4 route build --save,
    * uc4 route simulate .cache/last-bundle.json,
    * uc4 route exec .cache/last-bundle.json --yes,
    * uc4 route tail --from-block $(deployBlock) --max 20,
    * verify balances & logs.
71. Ensure deterministic JSON key ordering for bundles (stable diffs).
72. Add JSON schema $id & version field to bundle file; validate on load.
73. Gate exec on simulate.pass unless --ignore-sim-fail provided; default safe path: block.
74. Make --ignore-sim-fail print a scary confirmation unless --yes.
75. Add route decode-tx --hash 0x... to ABI-decode input data for debugging; uses executor ABI.
76. Add route receipts --address <executor> to fetch recent receipts (bounded) for operator audit.
77. Performance sanity: warm up provider with eth\_chainId and eth\_blockNumber before expensive calls.
78. Collapse multiple eth\_calls into a single multicall if chain supports (optional MVP).
79. Final CI hardening: flaky retry for tail-only tests (max 1 retry) with clear logs.
80. Prepare demo script scripts/demo/m6-demo.sh to showcase build→simulate→exec→tail in 90 seconds.

## **Post-merge hygiene (221–236)**

1. Update project board: move M6 cards to **Done**, link to PR and artifacts.
2. Open follow-ups: multicall batching, richer metrics, walletconnect integration (future).
3. Add CODEOWNERS coverage for packages/adapters-evm & packages/cli.
4. Ensure Dependabot has package-ecosystem: github-actions and npm for these packages.
5. Verify license headers across new files.
6. Add a short **video/gif** (optional) in README demonstrating CLI usage.
7. Validate reproducibility on clean machine/container with only M0 prerequisites installed.
8. Run secret scan locally (gitleaks) to confirm no accidental key commits.
9. Confirm pnpm -w build && pnpm -w test succeeds on branch without any local state.
10. Re-run the smoke script; archive its outputs into artifacts/m6/.

## **Final sign-off checklist (237–260)**

1. ✅ All CLI commands return non-zero exit codes on failure and clear messages.
2. ✅ UniswapV3AdapterClient integrated, unit-tested, documented.
3. ✅ E2E local path replicable; single command smoke works.
4. ✅ CI green on Node 18 & 20; artifacts uploaded.
5. ✅ Docs cover operator runbooks and troubleshooting.
6. ✅ Example bundles produce stable hashes; inspect clarity is high.
7. ✅ Safety guards: slippage clamp, deadline clamp, chainId check, recipient validation.
8. ✅ Error decoder maps custom errors reliably; raw data fallback documented.
9. ✅ Metrics/report (optional) works; off by default.
10. ✅ Config resolution robust; actionable messages on missing addresses.
11. ✅ Coverage meets targets; flaky tests addressed.
12. ✅ Version bump + tag + changelog committed.
13. ✅ Stakeholder demo recorded or notes attached.
14. ✅ Merge PR; close M6 milestone.

251–260. **Reserved** buffer (use if you add tiny glue tasks during implementation; keep total ≤ 260).