# **PR: [M5] TypeScript Core — Builders, Types, Serializers, Parity Tests**

**Branch:** feature/m5-ts-core-builders  
 **Targets:** UC-4 MVP core (feeds M6 CLI, M7 tests, M8 observability)  
 **Scope (high level):**

* Strong TS types mirroring on-chain structs (Bundle, Step, Policy)
* Uniswap V3 path encoder/decoder (exact ABI layout)
* Step & Bundle builders (with slippage clamp + deadline rules)
* Quoter integration (Uniswap V3 Quoter V2) for per-hop estimation
* Policy mirror checks (max hops, token & fee allowlists, deadline window)
* ABI serializers for Solidity parity (golden vectors)
* Thin contract clients (Executor, Vault, Adapter, Registry)
* Submit wrapper (optionally DRY-RUN/simulate)
* Unit + integration tests + golden fixtures
* CI wiring (typecheck, tests, coverage artifacts)

1. How to pull this PR locally

git checkout -b feature/m5-ts-core-builders

# apply changes below

git add -A

git commit -m "M5: TS core builders, types, serializers, parity tests"

git push origin feature/m5-ts-core-builders

1. Config deltas

**config/policies.json** (add client-side mirror knobs; safe defaults)

{

"slippageBpsMax": 100,

+ "deadlineMaxDelaySecs": 300,

+ "maxHops": 3,

+ "allowedFeeTiers": [500, 3000, 10000],

+ "quoting": { "timeoutMs": 1500, "retry": { "attempts": 2, "backoffMs": 250 } }

}

**config/venues.json** (ensure univ3 endpoints present for M5)

{

"uniswapV3": {

- "quoter": "<addr>",

- "router": "<addr>"

+ "quoter": "<addr>",

+ "router": "<addr>",

+ "factory": "<addr>"

}

}

**.env.example** (developer ergonomics)

RPC\_URL=http://127.0.0.1:8545

CHAIN\_ID=31337

EXECUTOR\_ADDRESS=0xExecutor...

VAULT\_ADDRESS=0xVault...

UNIV3\_QUOTER=0xQuoter...

UNIV3\_ROUTER=0xRouter...

POLICY\_JSON=./config/policies.json

VENUES\_JSON=./config/venues.json

1. New package: packages/core-exec

**File tree (added):**

packages/core-exec/

package.json

tsconfig.json

src/

index.ts

BundleTypes.ts

featureFlags.ts

version.ts

ids.ts

abi/

Executor.json

SettlementVault.json

UniswapV3Adapter.json

QuoterV2.json

Registry.json

index.ts

config/

env.ts

addresses.ts

index.ts

internal/

assert.ts

units.ts

retry.ts

time.ts

log.ts

clients/

provider.ts

executor.ts

vault.ts

adapter.univ3.ts

registry.ts

quoter.univ3.ts

venues/univ3/

path.ts

fees.ts

quote.ts

validate.ts

policy/

mirror.ts

build/

steps.ts

bundle.ts

serialize.ts

execute/

submit.ts

schema/

step.schema.json

bundle.schema.json

validate.ts

format/

summary.ts

result.ts

run/

localExecute.ts

test/

fixtures/

univ3.path.json

step.hex.json

bundle.hex.json

unit/

path.spec.ts

steps.spec.ts

bundle.spec.ts

mirror.spec.ts

integration/

quote.univ3.fork.spec.ts

build.bundle.fork.spec.ts

policy.rejections.spec.ts

1. Key implementations (abridged)

**src/BundleTypes.ts** — single source of truth for TS types aligned with Solidity:

export type Address = `0x${string}`;

export type Hex = `0x${string}`;

export interface StepParams {

tokenIn: Address;

tokenOut: Address;

amountIn: bigint;

amountOutMin: bigint;

recipient: Address;

deadline: bigint;

path: Hex; // exact Uniswap V3 20/3/20... layout

payer?: Address;

feeOnTransfer?: boolean;

}

export interface Bundle {

venueId: string; // "UNIV3" for M5

steps: Hex[]; // ABI-encoded StepParams[]

deadline: bigint;

minTotalOut: bigint;

tokenIn: Address;

tokenOut: Address;

recipient: Address;

}

export interface RoutePolicy {

maxHops: number;

allowedFeeTiers: number[];

slippageBpsMax: number;

deadlineMaxDelaySecs: number;

}

**src/venues/univ3/path.ts** — exact encoder/decoder & golden vectors:

import { Address, Hex } from "../..";

export function encodePath(tokens: Address[], fees: number[]): Hex {

if (tokens.length !== fees.length + 1) throw new Error("PATH\_SHAPE");

const chunks: string[] = [];

for (let i = 0; i < fees.length; i++) {

chunks.push(tokens[i].toLowerCase().slice(2)); // 20 bytes

chunks.push(fees[i].toString(16).padStart(6, "0")); // 3 bytes

}

chunks.push(tokens[tokens.length - 1].toLowerCase().slice(2));

return `0x${chunks.join("")}` as Hex;

}

**src/venues/univ3/quote.ts** — Quoter V2 per-hop chaining:

import { PublicClient } from "viem";

import { Address } from "../..";

import { encodePath } from "./path";

export async function quoteExactInput(

client: PublicClient,

quoter: Address,

tokens: Address[],

fees: number[],

amountIn: bigint

): Promise<bigint> {

const path = encodePath(tokens, fees);

const data = await client.readContract({

address: quoter,

abi: /\* QuoterV2 ABI \*/ [] as any,

functionName: "quoteExactInput",

args: [path, amountIn]

});

// V2 returns (amountOut, ...), normalize to bigint

const [amountOut] = data as unknown as [bigint];

return amountOut;

}

**src/policy/mirror.ts** — fast-fail client-side policy checks:

import { RoutePolicy, Address } from "..";

export function clampSlippageBps(bps: number, policy: RoutePolicy) {

if (bps < 0) return 0;

return Math.min(bps, policy.slippageBpsMax);

}

export function assertMaxHops(hops: number, policy: RoutePolicy) {

if (hops > policy.maxHops) throw new Error(`E\_MAX\_HOPS:${hops}>${policy.maxHops}`);

}

export function assertFeeTiers(fees: number[], policy: RoutePolicy) {

for (const f of fees) if (!policy.allowedFeeTiers.includes(f)) throw new Error(`E\_FEE:${f}`);

}

export function assertDeadline(deadlineSec: number, nowSec: number, policy: RoutePolicy) {

if (deadlineSec - nowSec > policy.deadlineMaxDelaySecs)

throw new Error(`E\_DEADLINE\_WINDOW:${deadlineSec - nowSec}>${policy.deadlineMaxDelaySecs}`);

}

**src/build/steps.ts** — Step builder with quotes and minOut:

import { StepParams, Address, Hex, RoutePolicy } from "..";

import { encodePath } from "../venues/univ3/path";

import { quoteExactInput } from "../venues/univ3/quote";

import { clampSlippageBps, assertMaxHops, assertFeeTiers, assertDeadline } from "../policy/mirror";

export async function buildUniv3Steps(opts: {

client: any; quoter: Address; tokens: Address[]; fees: number[]; amountIn: bigint;

recipient: Address; slippageBps: number; deadlineSec: number; nowSec: number; policy: RoutePolicy;

}): Promise<{ steps: StepParams[]; amounts: bigint[]; }> {

const { client, quoter, tokens, fees, amountIn, recipient, slippageBps, deadlineSec, nowSec, policy } = opts;

assertMaxHops(fees.length, policy);

assertFeeTiers(fees, policy);

assertDeadline(deadlineSec, nowSec, policy);

const clamped = clampSlippageBps(slippageBps, policy);

// quote entire path once (exactInput across multi-hop)

const expectedOut = await quoteExactInput(client, quoter, tokens, fees, amountIn);

const minOut = (expectedOut \* BigInt(10\_000 - clamped)) / 10\_000n;

const path = encodePath(tokens, fees);

const step: StepParams = {

tokenIn: tokens[0],

tokenOut: tokens[tokens.length - 1],

amountIn,

amountOutMin: minOut,

recipient,

deadline: BigInt(deadlineSec),

path

};

return { steps: [step], amounts: [expectedOut] };

}

**src/build/bundle.ts** — Bundle assembly + summary:

import { Bundle, StepParams, Hex, Address } from "..";

import { serializeSteps } from "./serialize";

export function buildBundle(input: {

venueId: string; steps: StepParams[]; tokenIn: Address; tokenOut: Address;

minTotalOut: bigint; recipient: Address; deadlineSec: number;

}): { bundle: Bundle; stepsEncoded: Hex[] } {

const stepsEncoded = serializeSteps(input.steps);

const bundle: Bundle = {

venueId: input.venueId,

steps: stepsEncoded,

deadline: BigInt(input.deadlineSec),

minTotalOut: input.minTotalOut,

tokenIn: input.tokenIn,

tokenOut: input.tokenOut,

recipient: input.recipient

};

return { bundle, stepsEncoded };

}

**src/build/serialize.ts** — ABI parity with Solidity:

import { Hex, StepParams } from "..";

import { encodeAbiParameters, parseAbiParameters } from "viem";

const stepAbi = parseAbiParameters(

"address tokenIn, address tokenOut, uint256 amountIn, uint256 amountOutMin, address recipient, uint256 deadline, bytes path, address payer, bool feeOnTransfer"

);

export function serializeStep(p: StepParams): Hex {

return encodeAbiParameters(stepAbi, [

p.tokenIn, p.tokenOut, p.amountIn, p.amountOutMin, p.recipient, p.deadline, p.path,

p.payer ?? "0x0000000000000000000000000000000000000000", !!p.feeOnTransfer

]) as Hex;

}

export function serializeSteps(arr: StepParams[]): Hex[] {

return arr.map(serializeStep);

}

**src/execute/submit.ts** — DRY-RUN or send:

import { PublicClient, WalletClient } from "viem";

import { Address, Bundle } from "..";

export async function submitBundle(

publicClient: PublicClient,

walletClient: WalletClient,

executor: Address,

abi: any,

bundle: Bundle,

{ dryRun = true }: { dryRun?: boolean } = {}

) {

if (dryRun) {

// simulate for revert reason & gas

const result = await publicClient.simulateContract({

address: executor, abi, functionName: "executeBundle", args: [bundle]

});

return { simulated: true, request: result.request, gas: result.request.gas };

}

const hash = await walletClient.writeContract({ address: executor, abi, functionName: "executeBundle", args: [bundle] });

return { simulated: false, txHash: hash };

}

1. CLI wiring (minimal hooks; full CLI in M6)

**packages/cli/src/commands/route.ts** (new subcommands stubbed to call core-exec):

program

.command('build-univ3')

.requiredOption('--tokens <csv>')

.requiredOption('--fees <csv>')

.requiredOption('--amount-in <wei>')

.option('--slippage-bps <n>', 'default: from policies.json')

.option('--deadline-sec <n>', 'default: now+policy.deadlineMaxDelaySecs')

.action(buildUniv3Handler);

(Handlers call core-exec buildUniv3Steps + buildBundle and print JSON; full CLI features arrive in M6.)

1. Tests & fixtures

**Golden fixtures:** test/fixtures/{univ3.path.json,step.hex.json,bundle.hex.json} produced by a deterministic vector (addresses + fees + amounts).

**Unit tests (vitest):**

* path.spec.ts — known vectors encode/decode roundtrip
* steps.spec.ts — slippage clamp, deadline window, fee tier validation
* bundle.spec.ts — serialization roundtrip → Solidity decoder hex (fixture parity)
* mirror.spec.ts — policy rejections (E\_MAX\_HOPS, E\_FEE, E\_DEADLINE)

**Integration (optional fork):**

* quote.univ3.fork.spec.ts — Quoter V2 returns monotonically increasing out for larger amountIn across same path
* build.bundle.fork.spec.ts — Build → simulate on executor ABI (if deployed on fork)
* policy.rejections.spec.ts — Ensure bad fee tier or hop count fails before RPC calls

**Example test (abridged):**

import { encodePath } from "../../src/venues/univ3/path";

it("encodes two-hop path correctly", () => {

const t = ['0x1111111111111111111111111111111111111111','0x2222222222222222222222222222222222222222','0x3333333333333333333333333333333333333333'] as const;

const fees = [500, 3000];

expect(encodePath([...t], fees))

.toBe("0x1111...<20 bytes>01f4<3 bytes>2222...<20 bytes>0bb8<3 bytes>3333...<20 bytes>");

});

1. CI updates

**.github/workflows/ci.yml** (add jobs)

* ts-core-lint — eslint + prettier check
* ts-core-typecheck — pnpm --filter @project/core-exec typecheck
* ts-core-test — pnpm --filter @project/core-exec test --coverage (upload coverage + golden fixtures)
* Cache pnpm store; cache node\_modules/.pnpm keyed by lockfile; upload dist/ as artifact

1. Rollout (local → fork)
2. **Local (anvil):** build + unit tests pass (no RPC needed).
3. **Fork (optional):** set FORK\_URL + real UNIV3\_QUOTER/ROUTER → run integration tests; record outputs.
4. **Consumers:** M6 CLI branch points to @project/core-exec APIs (no breaking changes expected).
5. Backward compatibility

* No contracts changed in M5; safe to merge independently.
* If policies.json lacks new fields, defaults applied by config/env.ts (slippage max=100, deadline=300s, maxHops=3).
* venues.json must include uniswapV3.quoter; otherwise quoting is disabled gracefully with a descriptive error.

1. Risks & mitigations

* **ABI drift vs Solidity:** golden vectors + roundtrip serializer tests in CI.
* **RPC variance:** retry/backoff helper in quotes (configurable) + unit tests don’t rely on RPC.
* **Amount precision:** all amounts are bigint; helpers forbid Number math; TS config exactOptionalPropertyTypes enabled.

1. Acceptance (Definition of Done)

* ✅ @project/core-exec builds and exports: BundleTypes, buildUniv3Steps, buildBundle, serializeSteps, submitBundle.
* ✅ Golden vectors pass; Solidity decoder (where available) accepts encoded steps.
* ✅ Unit coverage ≥ **90%** for venues/univ3/\* and build/\*.
* ✅ pnpm --filter @project/core-exec test green locally and on CI.
* ✅ Example smoke (local): build a 2-hop bundle JSON with clamped slippage and coherent minTotalOut.

1. Commit plan (squashable)
2. feat(core-exec): scaffold package, strict tsconfig, exports
3. feat(core-exec): BundleTypes and ABI serializers (golden vectors)
4. feat(core-exec): univ3 path encoder/decoder + tests
5. feat(core-exec): policy mirror checks (maxHops, fees, deadline)
6. feat(core-exec): quotes via Quoter V2 + retry/backoff
7. feat(core-exec): step & bundle builders + summary
8. feat(core-exec): submit wrapper (simulate / send)
9. test(core-exec): unit & integration + fixtures
10. ci: add ts-core jobs, cache, coverage artifacts
11. docs: M5 stubs (usage & examples)
12. Snippets for quick start (optional)

**Install & build**

pnpm -w i

pnpm --filter @project/core-exec build

pnpm --filter @project/core-exec test

**Programmatic usage**

import { buildUniv3Steps, buildBundle } from "@project/core-exec";

const { steps, amounts } = await buildUniv3Steps({

client, quoter, tokens, fees, amountIn, recipient, slippageBps: 80,

deadlineSec: Math.floor(Date.now()/1000)+300, nowSec: Math.floor(Date.now()/1000),

policy: { maxHops:3, allowedFeeTiers:[500,3000,10000], slippageBpsMax:100, deadlineMaxDelaySecs:300 }

});

const { bundle } = buildBundle({

venueId: "UNIV3",

steps, tokenIn: tokens[0], tokenOut: tokens.at(-1)!, minTotalOut: (amounts[0]\*99n)/100n,

recipient, deadlineSec: Math.floor(Date.now()/1000)+300

});