ANASTASIA LABS

Lucid Evolution 2.0

Proof of Achievement - Milestone 2

Project Id	1300126
Project Manager	Jonathan Rodriguez
Proposal Link	Catalyst Proposal

Contents

Blueprint & Enhanced Plutus Schema

Safe Deserialization and Type-Safe Derivation

1. Introduction

The second milestone of Lucid Evolution 2.0 (Blueprint & Enhanced Plutus Schema) focuses on implementing **safe deserialization** for UTxO datums and establishing an **automatic**, **type-safe derivation framework** for Datum and Redeemer types based on CIP-57 blueprints. This report summarizes the design decisions, development work, testing activities, and deliverables that ensure robust runtime validation and seamless compatibility with Plutus data structures and Aiken contracts.

2. Objectives and Acceptance Criteria

2.1. Objectives

- **Safe Deserialization:** Develop utilities to securely deserialize UTxO datum fields, leveraging Effect Schema for runtime integrity checks.
- **Type-Safe Derivation Framework:** Implement automatic derivation of Datum and Redeemer TypeScript types from Plutus blueprint files, ensuring full alignment with CIP-57 standards.
- **Practical Demonstrations:** Provide concrete examples, both code and video, showcasing the derivation process within mocked Aiken contracts.

2.2. Acceptance Criteria

- **Correct Derivation:** The framework must accurately derive Datum and Redeemer types from a mocked Aiken contract specification.
- **Runtime Validation:** Tests must validate type safety at runtime, producing descriptive error messages on invalid inputs.
- **Error Handling:** Invalid or malformed datum inputs must trigger clear, actionable error outputs.
- **Video Walkthrough:** A demonstration video must illustrate practical derivation examples and runtime checks.

3. Implementation Overview

3.1. Safe Deserialization of UTxO Datums

Effect Schema Integration: Utilized Effect Schema combinators to define datum-parsing schemas that enforce both structural correctness and type constraints.

Error Reporting: Custom error constructs provide detailed context field names, expected types, and actual values when deserialization fails.

Round-trip Guarantees: Ensured that deserialized datums, once re-serialized, match their original on-chain byte representations.

3.2. Type-Safe Derivation Framework

Blueprint Parsing Engine: Built a parser to read CIP-57-compliant blueprint JSON/YAML files and auto-generate corresponding TypeScript interfaces for Datum and Redeemer.

Compile-time & Runtime Alignment: Employed TypeScript generics and Effect Schema to synchronize compile-time types with runtime validators.

Mocked Aiken Contract Examples: Developed sample Aiken contracts demonstrating how on-chain parameters translate into derived types—covered in accompanying video.

Developer Workflow: Designed CLI commands for blueprint ingestion (derive-datums) and integration into existing Lucid transaction builders.

4. Evidence of Milestone Completion

The following items have been provided in the Lucid Evolution GitHub repository (https://github.com/Anastasia-Labs/lucid-evolution) as evidence of the successful completion of Milestone 1:

4.1. Implementation Code & Documentation

The implementation of the safe deserialization of UTxO datums, can be found at:

4.2. Unit Testing

The unit tests for the safe deserialization of UTxO datums can be found at:

4.3. Practical Examples & Video Demonstration

- **Mocked Aiken Contracts**
  - The mocked Aiken contracts demonstrating the type-safe derivation framework are located in:
- **Demonstration Video**
  - The demonstration video showcasing the derivation process and runtime checks is available at:

    docs/Milestone2/derivation-demo.mp4 (5:46 mins) illustrates:
  - Running lucid-schema derive.
  - Importing generated code into a Lucid off-chain script.
  - Observing both successful and failing deserialization in action.

5. Conclusion and Next Steps

5.1. Conclusion

Milestone 2 has been successfully delivered. We have established a robust, secure mechanism for safe deserialization of on-chain datums and an automated, type-safe framework for deriving Datum and Redeemer types from CIP-57 blueprints. All acceptance criteria have been met, with complete test coverage and practical demonstrations.

5.2. Next Steps

Advanced Features and Integration

Implement configurable encoding options (both bounded/canonical and unbounded/non-canonical), develop customizable data handling for specific datum components, integrate the schema package into the transaction builder, and enhance support for recursive Plutus types.

Utility Functions for Cardano Types

Create utility functions for converting between CBOR and key Plutus types (Address, Value, Credentials, OutputReference, CIP68 Metadata) and implement a comprehensive test suite for these functions.

Lucid Evolution Integration & Documentation

Deliver comprehensive, developer-friendly documentation aligned with Cardano standards, provide a detailed project closeout report, and produce a demonstration video highlighting the improvements in Lucid Evolution.