ANASTASIA LABS

Lucid Evolution 2.0

Proof of Achievement - Milestone 1

Project Id	1300126
Project Manager	Jonathan Rodriguez
Proposal Link	Catalyst Proposal

Contents

1. Executive Summary 1

2. Objectives and Acceptance Criteria 2
- 2.1. Objectives 2
- 2.2. Acceptance Criteria 2

3. Implementation Overview 4
- 3.1. Core Schema Development 4
- 3.2. Utility Functions and Type Safety 4
- 3.3. Documentation and Reporting 4

4. Evidence of Milestone Completion 6
- 4.1. GitHub Repository: 6
- 4.2. Unit Testing 6
- 4.3. Validation Report (PDF) 6

5. Conclusion and Next Steps 7
- 5.1. Conclusion 7
- 5.2. Next Steps 7

Blueprint & Enhanced Plutus Schema

1. Executive Summary

This first milestone of the Lucid Evolution 2.0 project is focused on implementing the core data schema structures using Effect Schema to support Plutus data types.

This report summarizes the design, implementation, testing, and documentation efforts aimed at providing a flexible, type-safe infrastructure for handling complex Cardano data types.

The work ensures that the core schema not only facilitates compile-time type safety through TypeScript integration but also guarantees runtime validation and reliable roundtrip encoding/decoding for all supported Plutus data types.

2. Objectives and Acceptance Criteria

2.1. Objectives

- **Implement Core Data Schemas:** Develop data schema structures utilizing Effect Schema that can handle various Plutus data types.
- **Utility Functions:** Build utility functions that ensure both compile-time and runtime type safety.
- **Testing Framework:** Create comprehensive unit tests covering roundtrip conversions, ensuring robust encoding and decoding functionality.
- **Documentation:** Produce detailed usage documentation with real-world examples to assist developers in integrating and leveraging the new schema.

2.2. Acceptance Criteria

- **Data Type Support:**

The schema must reliably support:

- `Integer`
- `ByteString`
- `List` (including nested lists)
- `Map`
- `Constr` (data with constructors)

- **Utility Functions Must:**

- Leverage Effect Schema for compile-time type safety validation of Plutus Data.
- Perform runtime type parsing to ensure data integrity.
- Successfully execute roundtrip (encode/decode) operations, validated by exhaustive tests.

- **Deliverables:**

- Implementation code in the Lucid Evolution GitHub repository.
- Comprehensive unit tests demonstrating bi-directional schema parsing for each data type.
- Documentation detailing schema usage with concrete examples.
- A PDF report showcasing Effect Schema's runtime validation.

3. Implementation Overview

3.1. Core Schema Development

Data Structure Implementation:

The development phase concentrated on creating schema structures that directly map to the requirements of the Plutus data types. The implementation includes:

- **Integer & ByteString:** Basic types are implemented with specific utility functions to manage encoding and decoding.
- **List Support:** Ability to manage both simple and nested lists, ensuring that recursive data structures are correctly handled.
- **Map and Constr:** Detailed handling for key-value pairs and constructors ensures that complex data combinations maintain integrity.

3.2. Utility Functions and Type Safety

Compile-time Assurance:

The development leverages TypeScript's type system in conjunction with Effect Schema to validate data types during the development stage, minimizing runtime errors.

Runtime Validation:

Custom utility functions were designed to parse and enforce Plutus Data integrity at runtime. This layer of validation protects against improper data handling during execution.

Roundtrip Testing:

A comprehensive suite of unit tests was implemented to verify bidirectional conversions:

- Encoding data structures into the Plutus format.
- Decoding back to the original format to confirm consistency.

3.3. Documentation and Reporting

Documentation:

Detailed documentation has been prepared, outlining:

- How to integrate and use the schema.
- Examples demonstrating type safety at compile time and runtime.
- Step-by-step guides on testing the roundtrip encoding/decoding process.

Evidence of Validation:

A demonstration PDF report has been generated, which showcases the Effect Schema's validation features and provides a clear narrative of the testing outcomes. The documentation and demonstration materials are all hosted on the GitHub repository.

4. Evidence of Milestone Completion

The following items have been provided as evidence of the successful completion of Milestone 1:

4.1. GitHub Repository:

The Lucid Evolution GitHub repository (https://github.com/Anastasia-Labs/lucid-evolution) contains:

- The full implementation of the core schema structures.
- The utility functions for compile-time and runtime validation.
- Extensive unit test coverage demonstrating the correct bi-directional parsing of Plutus Data types.
- Detailed documentation that includes usage examples and integration guidelines.

4.2. Unit Testing

A suite of unit tests has been executed which covers:

- Bi-directional schema parsing (to and from PlutusData conversions) for each supported data type.
- Roundtrip encoding and decoding processes.
- Runtime validation scenarios that ensure data integrity under various use cases.

4.3. Validation Report (PDF)

A comprehensive PDF report accompanies this milestone submission. This report showcases:

- Detailed test results.
- Example use cases.
- Visualizations of error handling that clearly demonstrate the efficacy of the runtime type parsing and validation logic implemented using Effect Schema.

5. Conclusion and Next Steps

5.1. Conclusion

Milestone 1 has been successfully achieved. The core schema structures and accompanying utility functions now provide robust support for critical Plutus data types, with assurance provided by rigorous compile-time and runtime validations. This foundation is essential for the subsequent enhancements and integrations planned for Lucid Evolution 2.0.

5.2. Next Steps

Safe Deserialization and Type-Safe Derivation

Automatically derive Datum and Redeemer types from Plutus blueprint files using practical examples, ensuring full compatibility with CIP-57 standards.

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.