ANASTASIA LABS

Proof of Achievement - Milestone 2

Highlighting Identified Gaps and Upgrades

Project Number	1100024
Project Manager	Jonathan Rodriguez

ANASTASIA LABS

Contents

Project Name: Lucid Evolution: Redefining Off-Chain Transactions in Cardano
URL: Catalyst Proposal

Introduction

Our short-term goal with Lucid Evolution isn't to reinvent the wheel but to make it better. We're focusing on handling side effects, improving error control, offering unsafe, safe, and lazy APIs, and providing safe deserialization schemas. We have implemented an extensive utility function variety and we aim to make it easier for maintainers.

In our following report for Milestone 2 we introduce our approach to the utility functions used in Lucid-evolution.

In this report, we briefly discuss our changes and highlight our progress, focusing on the gaps identified and addressed beyond the scope of our new utility packages detailed in the following report "Utility Functions"

Function Design / Gap Identification

After evaluating the legacy Lucid library and our initial implementations, we identified areas needing enhancements. In this effort, a big portion of the legacy library has been rewritten or created from scratch.

Lucid Evolution is like the legacy Lucid library but with improved APIs, better error handling, more structure, and the latest version of CML. Additionally, we're planning to introduce an abstraction layer on top, allowing users to select the serialization library that best suits their needs.

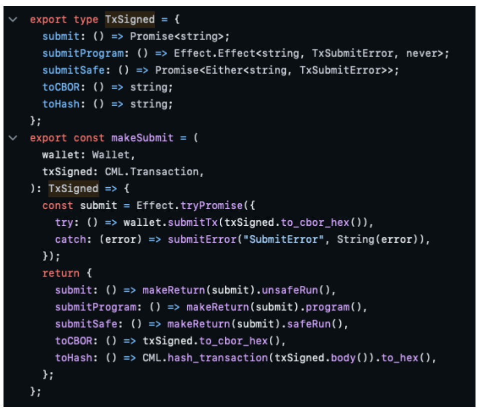
We restructured and refactored the legacy library to bring it up to date with today's resources. For example, we have a modified coinSelection algorithm, a new TxBuilder with its own function packages.

These can be grouped under

- Attach.ts
- Collect.ts
- CompleteTxBuilder.ts
- In work-Governance.ts
- Interval.ts
- Metadata.ts
- Mint.ts
- Pay.ts
- Pool.ts
- Read.ts
- Signer.ts
- Stake.ts
- TxUtils.ts

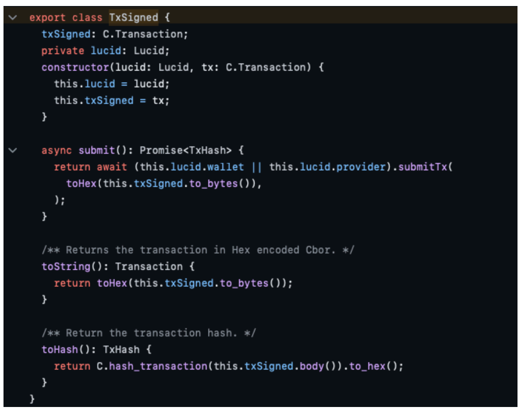
An example

In order to highlight differences between the evolution library and the legacy lucid library, we can display an example of how the two libraries handle the same transaction signed type in different syntaxes:



```
export type TxSigned = {
  submit: () => Promise<string>;
  submitProgram: () => Effect.Effect<string, TxSubmitError, never>;
  submitSafe: () => Promise<Either<string, TxSubmitError>>;
  toCBOR: () => string;
  toHash: () => string;
};
export const makeSubmit = (
  wallet: Wallet,
  txSigned: CML.Transaction,
): TxSigned => {
  const submit = Effect.tryPromise({
    try: () => wallet.submitTx(txSigned.to_cbor_hex()),
    catch: (error) => submitError("SubmitError", String(error)),
  });
  return {
    submit: () => makeReturn(submit).unsafeRun(),
    submitProgram: () => makeReturn(submit).program(),
    submitSafe: () => makeReturn(submit).safeRun(),
    toCBOR: () => txSigned.to_cbor_hex(),
    toHash: () => CML.hash_transaction(txSigned.body()).to_hex(),
  };
};
```

Figure 1: Lucid Evolution - TxSigned type



```
export class TxSigned {
  txSigned: C.Transaction;
  private lucid: Lucid;
  constructor(lucid: Lucid, tx: C.Transaction) {
    this.lucid = lucid;
    this.txSigned = tx;
  }

  async submit(): Promise<TxHash> {
    return await (this.lucid.wallet || this.lucid.provider).submitTx(
      toHex(this.txSigned.to_bytes()),
    );
  }

  /** Returns the transaction in Hex encoded Cbor. */
  toString(): Transaction {
    return toHex(this.txSigned.to_bytes());
  }

  /** Return the transaction hash. */
  toHash(): TxHash {
    return C.hash_transaction(this.txSigned.body()).to_hex();
  }
}
```

Figure 2: Lucid - TxSigned class

Highlights

We have adopted an approach closer to functional programming paradigms, using `Effect` to handle promises and improve error management, along with the latest version of `CML`.

As highlighted in our latest release patch notes (0.2.47), we are working on enhancing and upgrading the variability of tools and services available for developers using lucid evolution.

By upgrading the compatibility, we are broadening the library's reach to support various Cardano environments. We have addressed previous issues with TypeScript configuration, improving type declarations and enhancing provider variability through integration.

Transaction management has seen significant improvements, including sophisticated UTXO management that allows efficient chaining of transactions within a single block. Memory management was optimized minimizing memory leaks and enhancing overall stability by upgrading to the latest CML versions.

These enhancements, reflect our commitment to addressing gaps and improving the library. We are constantly in communication with our community through official social channels and actively discuss about ways to improve Lucid Evolution

We are progressing towards implementing Conway functions to support developers with governance era functionalities. Additionally, we are developing a refined, dedicated documentation website to help more developers onboard Lucid Evolution into their workflow. We are keeping pace with Cardano's evolution.

We aim to create a library that, just like our design patterns repository, simplifies complex design patterns and provides developers with an efficient, modern tool.

Use Case Scenario

As an example of how lucid evolution could be used for common design patterns, Here's how the Lucid Evolution enabled input indexing could look like, making Staking Validator Design Pattern an easy implementation

```
withdraw (
  rewardAddress: RewardAddress,
  amount: Lovelace,
  redeemer?: string | RedeemerBuilder,
) => TxBuilder;

// The type which needs to be provided in case you want your redeemer to
// have input indices but would like lucid to populate them for you
// after doing the coin selection
export type RedeemerBuilder = {
  makeRedeemer: (inputIndices: bigint[]) => Redeemer;
  inputs: UTxO[];
};

const rdmrBuilder: RedeemerBuilder = {
  makeRedeemer: (inputIndices: bigint[]) => {
    return Data.to({
    nodeIdxs: inputIndices,
    nodeOutIdxs: outputIndices, // you would have this already
  })},
  inputs: selectedUTxOs // any inputs that you wish to be indexed, the inputs
}

const tx = lucid_evolution.
  .newTx()
  .collectFrom(selectUTxOs, redeemer)
  .withdraw(rewardAddress, 0n, rdmrBuilder)
  .attach.SpendingValidator(spend)
  .attach.WithdrawalValidator(stake)
  .completeProgram();
```