ANASTASIA LABS

Proof of Achievement - Milestone 5

Bug Resolution and Community Requests

Project Number 1100024
Project Manager Jonathan Rodriguez

ANASTASIA LABS

Contents

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

Evidence Definition

Provided documentation of the steps taken to address each identified bug or issue, including code changes, patches, or updates implemented to resolve them is available

Bug Resolution and Community Requests

Throughout the development of Lucid Evolution, we've addressed numerous bugs, issues, and community requests, as evidenced by:

- 1050+ commits
- 435+ releases
- 55+ **closed** issues
- 345+ **closed** pull requests

These can be verified in our repository where we keep track of all discussions that take place about and around lucid evolution. Even though our primary communication channel with the developer community is the dedicated Discord channel, we still refer everyone to use github issues and PRs to sustainably maintain our open source library.

While documenting every bug fix would be impractical for this report, we'll highlight several representative examples that demonstrate our approach to:

1. Bug fixes
2. Performance improvements
3. Community-requested features

Links

Commits

https://github.com/Anastasia-Labs/lucid-evolution/commits/main/

Releases

https://github.com/Anastasia-Labs/lucid-evolution/releases

Github Issues

https://github.com/Anastasia-Labs/lucid-evolution/issues?q=is%3Aissue+is%3Aclosed

Pull Requests we handled

https://github.com/Anastasia-Labs/lucid-evolution/pulls?q=is%3Apr+is%3Aclosed

Example: Fee Calculation for Reference Scripts

Context

During Conway era updates on Cardano, a issue emerged regarding fee calculations when spending UTXOs containing reference scripts.

Challenge

- Transaction fees were being incorrectly calculated when spending UTXOs with attached reference scripts
- The issue became particularly relevant due to Conway era updates modifying fee structures
- Required careful consideration of both user experience and network rules

Technical Resolution

1. Identified that reference scripts now carry additional weight in fee calculations
2. Modified coin selection logic to handle reference script UTXOs differently
3. Implemented explicit fee handling for reference script transactions
4. Added testing to verify the solution

Impact for Developers

- More accurate fee estimation for transactions
- Improved handling of reference script UTXOs
- Better alignment with Conway era requirements

Related Links

- Issue: Fee calculation error 223
- Fix PR: Fix reference script fees issues 233
- Related CML Update: dcSpark/cardano-multiplatform-lib349

Example 2: Datum Handling Consistency in Contract Payments

Context

A compatibility issue was identified where datum handling behavior differed between Legacy Lucid and Lucid Evolution, potentially affecting dApp migrations and existing smart contract interactions.

Challenge

- Inconsistent datum inclusion behavior in witness sets between library versions
- Affected common `payToContract` operations with datum hashes
- Impacted developers migrating from legacy Lucid to Lucid Evolution
- Required maintaining backward compatibility while improving the codebase

Technical Resolution

1. Identified the behavioral difference in datum handling
2. Implemented automatic datum inclusion for `asHash` kind in payToContract operations
3. Released fix in version 0.3.10
4. Verified solution through community testing

Impact for Developers

- Consistent behavior with legacy Lucid library
- Simplified migration path for existing dApps
- Improved reliability in smart contract interactions
- Better backward compatibility

Related Links

- Issue: Inconsistent Datum Inclusion 227
- Fix PR: Fix datum inclusion for asHash kind 228
- Migration Guide: Migration guide from Lucid 189
- Release: v0.3.10

Example 3: Wallet Fund Management and Transaction Fee Optimization

Context

VESPR Wallet team, during their migration to Lucid Evolution, identified a critical functionality gap in handling scenarios where users want to send all their ADA funds. This highlighted our library's role in supporting production wallets and maintaining compatibility with existing user experiences.

Challenge

- Users couldn't send all their ADA when remaining funds were insufficient for change UTxO
- Previous behavior in legacy Lucid automatically added remaining amount to transaction fee
- Required maintaining wallet draining functionality while ensuring precise transaction control
- Needed to support multiple wallet implementation scenarios

Technical Resolution

1. Implemented new configuration option `includeLeftoverLovelaceAsFee`
2. Added flexibility in handling remaining funds:
   - Option to throw error (default behavior)
   - Option to include leftover in transaction fee
3. Maintained backward compatibility with existing implementations
4. Verified solution through direct testing with VESPR wallet team

Impact for Developers

- Simplified wallet draining operations
- More control over transaction fee handling
- Better support for production wallet implementations
- Improved migration path from legacy Lucid

Related Links

- Issue: Wallet Fund Management 368
- Fix PR: Add option to include tiny change 392
- Implementation: TxBuilder.ts

Example 4: Script Integrity Hash Validation

Context

A issue emerged where transactions using datum of kind “asHash” were failing with PPViewHashesDontMatch errors. Our library’s deep integration with Cardano’s protocol-level features and our coordination with other ecosystem tools have been shown through this example

Challenge

- Transactions with “asHash” datum type were failing validation
- Error specifically related to script integrity hashes
- Affected multiple versions (0.3.15 through 0.3.18)
- Required coordination with CML (Cardano Multiplatform Library) team

Technical Resolution

1. Identified root cause in script integrity hash calculation
2. Coordinated with dcSpark team on CML updates
3. Implemented fix through CML version bump
4. Verified solution across different transaction scenarios

Impact for Developers

- Restored reliable datum hash handling
- Conway era compatibility
- Transaction validation reliability
- Seamless integration with latest protocol features

Related Links

- Issue: Script Integrity Hash Issue 261
- CML Fix: dcSpark/cardano-multiplatform-lib351
- Implementation: Bump CML version 285

Example 5: Build System Enhancement for Developer Testing

Context

A community developer identified challenges in the build distribution structure that affected the testing workflow. This highlighted our commitment to improving developer experience and maintaining an efficient development cycle.

Challenge

- Build process distributed packages under complex file structures
- Developers faced difficulties in quickly testing new versions
- Needed streamlined process for local development and testing
- Required clear documentation for package management

Technical Resolution

1. Clarified build process documentation
2. Implemented package-specific build commands
3. Added support for quick testing via `pnpm pack`
4. Created streamlined workflow for local development:
   - Package-level building
   - Tarball generation
   - Local dependency integration

Impact for Developers

- Simplified local development workflow
- Faster testing cycles
- Clearer package management process
- Improved contribution experience

Related Links

- Issue: Build Distribution Structure 140
- NPM Package: lucid-evolution/lucid
- Test Directory: Test Examples

Example 6: Hardware Wallet Compatibility and CIP-21 Compliance

Context

A compatibility issue was identified with Trezor hardware wallet transactions, highlighting the importance of adhering to Cardano protocol standards (CIP-21) and maintaining broad hardware wallet support.

Challenge

- Transactions were failing specifically on Trezor hardware wallets
- Root cause identified as non-compliance with CIP-21 map ordering requirements
- Affected transactions with datums required special handling
- Needed solution compatible with hardware wallet security models

Technical Resolution

1. Implemented canonical format for transaction data structures
2. Added proper map ordering according to CIP-21 specifications
3. Enhanced compatibility with cardano-hw-cli transformations
4. Released fix in version 0.3.37 with comprehensive changes

Impact for Developers

- Improved hardware wallet support
- Better compliance with Cardano standards
- Enhanced transaction reliability
- Simplified hardware wallet integration

Related Links

- Issue: Hardware Wallet Compatibility 196
- Fix PR: Implement canonical format 333
- CIP-21 Standard: Cardano Transaction Metadata Format
- Release: v0.3.37

Example 7: Transaction Signer Flexibility

Context

The community identified a limitation in the transaction signing API where developers needed more flexible ways to add signers to transactions, particularly when working with key hashes directly. This showcased our responsive approach to developer needs and API enhancement.

Challenge

- Original API only accepted Address or RewardAddress types
- Developers sometimes had direct access to key hashes
- Needed to maintain backward compatibility
- Required consideration of different signing scenarios

Technical Resolution

1. Implemented two distinct methods for maximum flexibility:
   - `addSigner`: Handles Address and RewardAddress inputs
   - `addSignerKey`: Accepts PaymentKeyHash and StakeKeyHash inputs
2. Maintained backward compatibility
3. Enhanced type safety through explicit method separation
4. Provided comprehensive documentation for both approaches

Impact for Developers

- More flexible transaction signing options
- Clearer API semantics
- Better support for various implementation scenarios
- Simplified migration from legacy Lucid

Related Links

- Issue: Enhanced Signer Flexibility 265
- Implementation PR: Add signer by reward address 272
- Related Issue: Transaction submission 171

Example 8: Transaction Composition Enhancement

Context

The community, particularly Liqwid Labs and other DeFi projects, identified issues with the transaction composition functionality, a critical feature used in 90% of their transaction workflow code. This highlighted the importance of maintaining key features while improving implementation.

Challenge

- Transaction composition not functioning as expected
- Need to maintain composability for DeFi integrations
- Balance between clean code and practical functionality
- Required support for complex transaction scenarios

Technical Resolution

1. Reimplemented transaction composition with immutability focus
2. Enhanced state management for TxBuilderConfig components:
   - Proper merging of UTxO sets
   - Script deduplication
   - Fee calculation improvements
3. Maintained backward compatibility
4. Added comprehensive testing for composition scenarios

Impact for Developers

- Reliable transaction composition for DeFi integrations
- Cleaner, more predictable transaction building
- Better support for complex smart contract interactions
- Improved composability with external SDKs

Related Links

- Issue: Transaction Composition Fix 277
- Fix PR: Implement compose functionality 397
- Implementation: TxBuilder.ts

Example 9: Seed Phrase Generation and Wallet Creation

Context

During community migration from legacy Lucid to Lucid Evolution, a critical issue was identified in the wallet creation process using seed phrases.

Challenge

- Seed phrase generation failing with crypto hash finalization error
- Inconsistency between legacy and Evolution implementations
- Critical functionality affecting wallet creation workflows

Technical Resolution

1. Identified root cause in SHA256 hash implementation:
   - Legacy Lucid: Using Deno's standard hash library
   - Evolution: Using Node's crypto module
2. Implemented proper hash instance management
3. Added hash state reset functionality
4. Provided comprehensive testing for seed generation scenarios

Impact for Developers

- Reliable seed phrase generation
- Consistent wallet creation process
- Smooth migration path from legacy Lucid
- Improved cryptographic operation handling

Related Links

- Issue: Seed Generation Error 55
- Fix PR: Fix generateSeedPhrase 63
- Related Issue: Fix generateSeedPhrase error 56

Example 10: Stake Withdrawal Encoding Compatibility

Context

A change in stake withdrawal encoding was identified that affected interactions with existing Plutus V2 scripts.

Challenge

- Script purpose encoding changed between versions
- Affected existing Plutus V2 script interactions
- Required maintaining backward compatibility
- Needed coordination with Aiken compiler updates

Technical Resolution

1. Identified encoding change impact:
   - Pre v0.3.20: Double-wrapped encoding (121[122[...]])
   - Post v0.3.22: Single-wrapped encoding (122[...])
2. Investigated root cause in UPLC dependencies
3. Coordinated with Aiken team on wrapper changes
4. Released fix through UPLC version update

Impact for Developers

- Restored compatibility with existing scripts
- Maintained consistent withdrawal behavior
- Simplified script purpose encoding
- Improved integration with Aiken ecosystem

Related Links

- Issue: Stake Withdrawal Encoding 311
- Fix PR: Aiken UPLC Update 337
- Related Change: Aiken Wrapper Update