



ANASTASIA LABS

Proof of Achievement – Milestone 2

Augmenting Lucid's Utility Library Functions

Project Number 1100024

Project Manager Jonathan Rodriguez

Contents

Introduction	1
Function Design / Gap Identification	1
Outline Report for Utility Functions per Package	2
bip39.ts	3
address.ts	4
cbor.ts	5
cost_model.ts	6
credential.ts	7
datum.ts	8
keys.ts	9
native.ts	10
network.ts	11
scripts.ts	12
time.ts	13
utxo.ts	14
value.ts	15
Use Case Scenario	16
Testing Suite	17
Packages	17
GIF Testrun	17

Project Name: Lucid Evolution: Redefining Off-Chain Transactions in Cardano
URL: [Catalyst Proposal](#)

Introduction

Function Design / Gap Identification

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 also aim to make it easier for maintainers.

We plan to move away from CML in the long run, but it's not a big concern right now because it works. Most transaction builders stick with it since it's the most reliable serialization library out there.

We're also planning to offer an abstraction layer on top so people can choose whichever serialization library works best for them. We're taking a pragmatic approach, so it doesn't matter to us which library is used as long as it gets the job done.

Lucid Evolution is like legacy Lucid but with improved APIs, better error handling, more structure, and of course the latest version of CML.

Outline Report for Utility Functions per Package

In this following section you will find the utility packages we have under the [lucid-evolution github](#) page with the following general format:

1. Title
2. Description
3. Key Functions
4. Code Snapshot

Description

The `bip39.ts` module implements functions related to BIP39, which defines a way to generate the mnemonic phrase (a series of easy-to-remember words) from a random seed

- This is a partial reimplementation of [BIP39 in Deno](#)
- We only use the default Wordlist (english)

Utility Package	Directory
bip39.ts	GitHub Link

Key Functions

`mnemonicToEntropy`

Converts a mnemonic phrase back into its original entropy representation

`generateMnemonic`

Generates a new mnemonic phrase from random entropy. It can be used to create new wallets or regenerate existing ones from a known entropy source

`entropyToMnemonic`

Converts entropy into a mnemonic phrase using a specific wordlist for wallet recovery or setup

Code Snapshot

```

lucid-evolution / packages / bip39 / src / bip39.ts
Code Blame 2289 lines (2193 loc) · 27.6 KB
16 export function mnemonicToEntropy(
17   phrase: string
18 ): Uint8Array {
19   throw new Error(INVALID_CHECKSUM);
20 }
21 return toHex(entropy);
22 }
23 }
24
25 > function randomBytes(size: number): Uint8Array {
26   // ...
27 }
28
29 > export function generateMnemonic(
30   entropy: Uint8Array,
31   wordlist?: Array<string>,
32 ): string {
33   wordlist = wordlist || DEFAULT_WORDLIST;
34   if (!wordlist) {
35     throw new Error(WORDBLIST_REQUIRED);
36   }
37   // 128 <= ENT <= 256
38   if (entropy.length < 16) {
39     throw new TypeError(INVALID_ENTROPY);
40   }
41   if (entropy.length > 32) {
42     throw new TypeError(INVALID_ENTROPY);
43   }
44   if (entropy.length % 4 !== 0) {
45     throw new TypeError(INVALID_ENTROPY);
46   }
47   const entropyBits = bytesToBinary(Array.from(entropy));
48   const checksumBits = deriveChecksumBits(entropy);
49   const bits = entropyBits + checksumBits;
50   const chunks = bits.match(/.{1,11}/g);
51   const words = chunks.map((binary) => {
52     const index = binaryToByte(binary);
53     return wordlist[index];
54   });
55   return wordlist[0] === "u3042u3044u3053u304fu3057u3093" // Japanese wordlist
56     ? words.join("u3080")
57     : words.join(" ");
58 }
59
60 > function deriveChecksumBits(entropyBuffer: Uint8Array): string {
61   // ...
62 }
63
64 > function lpad(str: string, padString: string, length: number): string {
65   // ...
66 }
67
68 function bytesToBinary(bytes: Array<number>): string {
69   return bytes.map((x) => lpad(x.toString(2), "0", 8)).join("");
70 }
71

```

Figure 1: Snapshot-01-BIP39

Description

The `address.ts` module is used to handle address-related operations within the Lucid Evolution library. Its functions allow the manipulation and conversion of various address types

Utility Package	Directory
address.ts	GitHub Link

Key Functions

`addressFromHexOrBech32`

Converts an address from either hexadecimal or Bech32 format to a CML Address object

`credentialToRewardAddress`

Converts a stake credential into a reward address

`validatorToRewardAddress`

Converts a validator (either a certificate or withdrawal validator) into a reward address using the script hash derived from the validator

`getAddressDetails`

Extracts and returns detailed information about various address types (Base, Enterprise, Pointer, Reward, Byron), including payment and stake credentials

Code Snapshot

```
lucid-evolution / packages / utils / src / address.ts
Code Blame 239 lines (229 loc) · 6.97 KB
41 }
42
43 export function validatorToRewardAddress(
44   network: Network,
45   validator: CertificateValidator | WithdrawalValidator,
46 ): RewardAddress {
47   const validatorHash = validatorToScriptHash(validator);
48   return CML.RewardAddress.new(
49     networkId(network),
50     CML.Credential.new_script(CML.ScriptHash.from_hex(validatorHash)),
51   )
52   .to_address()
53   .to_bech32(undefined);
54 }
55
56 /** Address can be in Bech32 or Hex. */
57 export function getAddressDetails(address: string): AddressDetails {
58   // Base Address
59   try {
60     // const parsedAddress = CML.BaseAddress.from_address(
61     //   addressFromHexOrBech32(address),
62     // );
63     const parsedAddress = CML.BaseAddress.from_address(
64       CML.Address.from_bech32(address),
65     );
66     const paymentCredential: Credential =
67       parsedAddress.payment().kind() === 0
68       ? {
69         type: "Key",
70         // hash: toHex(
71         //   parsedAddress.payment_cred().to_keyhash().to_bytes(),
72         // ),
73         hash: parsedAddress.payment().as_pub_key().to_hex(),
74       }
75       : {
76         type: "Script",
77         // hash: toHex(
78         //   parsedAddress.payment_cred().to_scripthash().to_bytes(),
79         // ),
80         hash: parsedAddress.payment().as_script().to_hex(),
81       };
82     const stakeCredential: Credential =
83       parsedAddress.stake().kind() === 0
84       ? // parsedAddress.stake_cred().kind() === 0
85         {
86         type: "Key",
87         hash: parsedAddress.stake().as_pub_key().to_hex(),
88         // hash: toHex(parsedAddress.stake_cred().to_keyhash().to_bytes()),
89       }
90       : {
91         type: "Script",
92         // hash: toHex(parsedAddress.stake_cred().to_scripthash().to_bytes()),
```

Figure 2: Snapshot-02-Address

Description

The `cbor.ts` module within Lucid Evolution deals with functionalities related to CBOR (Concise Binary Object Representation), specifically focusing on encoding and decoding operations that adhere to the CBOR standard as defined in RFC 7049

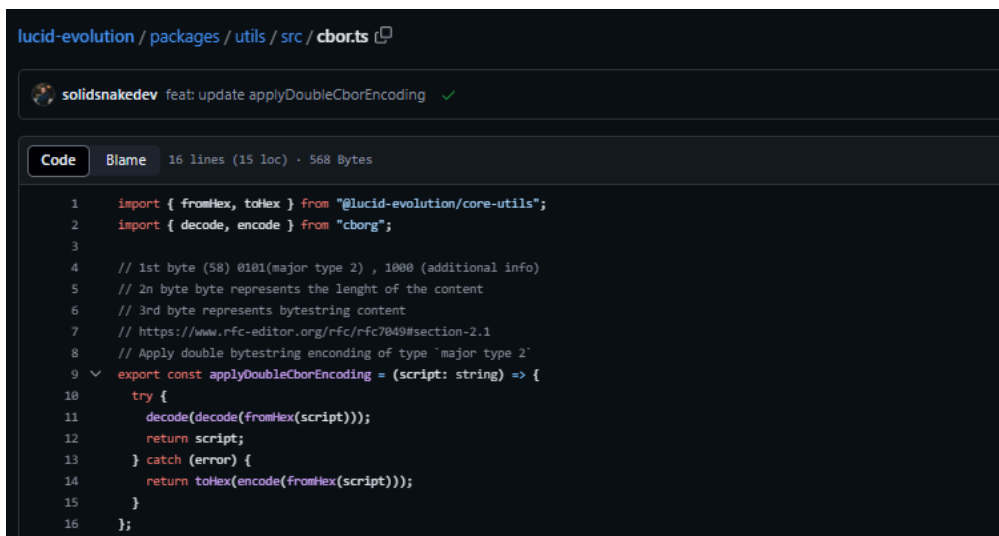
Utility Package	Directory
cbor.ts	GitHub Link

Key Functions

`applyDoubleCborEncoding`

Implements a double encoding for CBOR bytestrings, which decodes an encoded string twice to ensure correct formatting

Code Snapshot



```
lucid-evolution / packages / utils / src / cbor.ts
solidsnakedev feat: update applyDoubleCborEncoding ✓

Code Blame 16 lines (15 loc) · 568 Bytes

1  import { fromHex, toHex } from "@lucid-evolution/core-utils";
2  import { decode, encode } from "cborg";
3
4  // 1st byte (58) 0101(major type 2) , 1000 (additional info)
5  // 2n byte represents the length of the content
6  // 3rd byte represents bytestring content
7  // https://www.rfc-editor.org/rfc/rfc7049#section-2.1
8  // Apply double bytestring encoding of type 'major type 2'
9  export const applyDoubleCborEncoding = (script: string) => {
10   try {
11     decode(decode(fromHex(script)));
12     return script;
13   } catch (error) {
14     return toHex(encode(fromHex(script)));
15   }
16 }
```

Figure 3: Snapshot-03-Cbor

Description

The `cost_model.ts` module in Lucid Evolution deals with the configuration and management of cost models related to the execution of Plutus scripts on the blockchain. These cost models are used to determine the computational and memory costs of running smart contracts

Key Functions

`createCostModels`

Constructs a `CostModels` object that covers the various cost parameters for different versions of the Plutus scripts (PlutusV1, PlutusV2).

This function populates cost models from predefined settings.

1. Initializes new cost model objects for each Plutus version
2. Iteratively fills these objects with cost data parsed from input parameters
3. Handles the memory management of these operations to prevent leaks and ensure efficiency

Utility Package	Directory
<code>cost_model.ts</code>	GitHub Link

Code Snapshot

```

lucid-evolution/packages/utxos/src/cost_models
Code Blame 398 lines (392 loc) · 17.4 KB
4 export function createCostModels(costModels: CostModels): OM.CostModels {
5   const costmdis = OM.CostModels.new();
6
7   // add plutus v1
8   const costmdV1 = OM.IntList.new();
9   for (const cost of Object.values(costModels.PlutusV1)) {
10    const int = OM.Int.from_str(cost.toString());
11    costmdV1.add(int);
12    int.free();
13  }
14  costmdis.set_plutus_v1(costmdV1);
15  costmdV1.free();
16
17  // add plutus v2
18  const costmdV2 = OM.IntList.new();
19  for (const cost of Object.values(costModels.PlutusV2)) {
20    const int = OM.Int.from_str(cost.toString());
21    costmdV2.add(int);
22    int.free();
23  }
24  costmdis.set_plutus_v2(costmdV2);
25  costmdV2.free();
26
27  // add plutus v3
28  // const costmdV3 = C.IntList.new();
29  // Object.values(costModels.PlutusV3).forEach((cost) => {
30  //   costmdV3.add(C.Int.new(BigInt(cost)))
31  // });
32  // costmdis.set_plutus_v3(costmdV3);
33
34  return costmdis;
35 }
36
37 export const PROTOCOL_PARAMETERS_DEFAULT: ProtocolParameters = {
38   minFee: 44,
39   minFeeB: 155383,
40   maxTxSize: 16384,
41   maxValSize: 5000,
42   keyDeposit: 2000000n,
43   poolDeposit: 500000000n,
44   priceUtxo: 0.0577,
45   priceStep: 0.0000721,
46   maxTxExes: 1000000n,
47   maxTxExSteps: 1000000000n,
48   coinsPerUtxoByte: 4318n,
49   collateralPercentage: 150,
50   maxCollateralInputs: 3,
51   costModels: {
52     PlutusV1: {
53       "addInteger-cpu-arguments-intercept": 285665,
54       "addInteger-cpu-arguments-slope": 832,
55       "addInteger-memory-arguments-intercept": 1,

```

Figure 4: Snapshot-04-Costmodel

Description

The `credential.ts` module handles the creation and manipulation of credentials within the ecosystem. This module is for constructing addresses and managing their components.

Utility Package	Directory
credential.ts	GitHub Link

Key Functions

credentialToAddress

Converts payment and optionally stake credentials into an address

scriptHashToCredential

Wraps a script hash into a credential object, utilizing its use in other functions requiring a credential format

keyHashToCredential

Converts a key hash into a credential object, allowing for further operations that require credentials


paymentCredentialOf

Extracts the payment credential from an address, throwing an error if the address does not contain one

stakeCredentialOf

Retrieves the stake credential from a reward address

Code Snapshot



```

lucid-evolution/packages/utls/src/credential.ts
Code Blame 86 lines (81 loc) · 2.25 kB
5   ScriptHash,
6   KeyHash,
7   RewardAddress,
8   } from "lucid-evolution/core-types";
9   import { networkId } from "../network.js";
10  import { OM } from "../core.js";
11  import { getAddressDetails } from "../address.js";
12
13  export function credentialToAddress(
14    network: Network,
15    paymentCredential: Credential,
16    stakeCredential?: Credential,
17  ): Address {
18    if (stakeCredential) {
19      return OM.BaseAddress.new(
20        networkId(network),
21        paymentCredential.type === "Key"
22          ? OM.Credential.new_pub_key(
23              OM.Ed25519KeyHash.from_hash(paymentCredential.hash),
24            )
25          : OM.Credential.new_script(
26              OM.ScriptHash.from_hash(paymentCredential.hash),
27            ),
28        stakeCredential.type === "Key"
29          ? OM.Credential.new_pub_key(
30              OM.Ed25519KeyHash.from_hash(stakeCredential.hash),
31            )
32          : OM.Credential.new_script(
33              OM.ScriptHash.from_hash(stakeCredential.hash),
34            ),
35      ).to_address()
36        .to_bech32(undefined);
37    } else {
38      return OM.EnterpriseAddress.new(
39        networkId(network),
40        paymentCredential.type === "Key"
41          ? OM.Credential.new_pub_key(
42              OM.Ed25519KeyHash.from_hash(paymentCredential.hash),
43            )
44          : OM.Credential.new_script(
45              OM.ScriptHash.from_hash(paymentCredential.hash),
46            ),
47      ).to_address()
48        .to_bech32(undefined);
49    }
50  }
51
52  export function scriptHashToCredential(scriptHash: ScriptHash): Credential {
53    return {
54      type: "Script",
55      hash: scriptHash
56    };
57  }

```

Figure 5: Snapshot-05-Credential

Description

The `datum.ts` module provides functionality for handling Plutus data on the blockchain. Specifically, it includes utilities for converting Plutus data (datum) into a format that is suitable for transaction processing, like generating a hash of the datum

Utility Package	Directory
datum.ts	GitHub Link


Key Functions

datumToHash

Converts a datum object into its corresponding hash. This hash is used to refer to data stored off-chain.

1. Converts the datum from its CBOR hexadecimal representation to a Plutus data format
2. Uses the CML to calculate the hash of the Plutus data

Code Snapshot



```
lucid-evolution / packages / utils / src / datum.ts

solidsnakedev refactor: move CML to core file

Code Blame 6 lines (5 loc) • 232 Bytes

1 import { Datum, DatumHash } from "@lucid-evolution/core-types";
2 import { CML } from "./core.js";
3
4 export function datumToHash(datum: Datum): DatumHash {
5   return CML.hash_plutus_data(CML.PlutusData.from_cbor_hex(datum)).to_hex();
6 }
```

Figure 6: Snapshot-06-Datum

Description

The `keys.ts` deals with the generation and conversion of keys which are fundamental for secure transactions

Utility Package	Directory
keys.ts	GitHub Link

Key Functions

generatePrivateKey

Generates a new private key using the ED25519 cryptographic algorithm. This key is used for signing transactions securely

generateSeedPhrase

Creates a mnemonic seed phrase based on the BIP39 standard

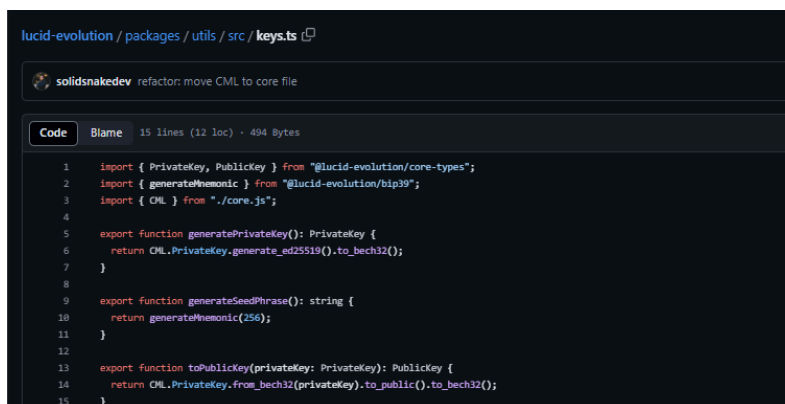
toPublicKey

Converts a given private key to its corresponding public key, allowing for the public key to be used in transaction verification without revealing the private key

An example of a key private -> public conversion would look like:

```
1 CML.PrivateKey.from_bech32(privateKey).to_public().to_bech32();
```

Code Snapshot



```
lucid-evolution / packages / utils / src / keys.ts
solidsnakedev refactor move CML to core file

Code Blame 15 lines (12 loc) · 494 Bytes

1 import { PrivateKey, PublicKey } from "@lucid-evolution/core-types";
2 import { generateMnemonic } from "@lucid-evolution/bip39";
3 import { CML } from "../core.js";
4
5 export function generatePrivateKey(): PrivateKey {
6   return CML.PrivateKey.generate_ed25519().to_bech32();
7 }
8
9 export function generateSeedPhrase(): string {
10   return generateMnemonic(256);
11 }
12
13 export function toPublicKey(privateKey: PrivateKey): PublicKey {
14   return CML.PrivateKey.from_bech32(privateKey).to_public().to_bech32();
15 }
```

Figure 7: Snapshot-07-Keys

Description

The `native.ts` module handles operations related to Cardano's native scripts, which are used for transaction validation without the execution of Plutus smart contracts. This module provides functionality to convert custom native script objects into Cardano's native script format

Utility Package	Directory
native.ts	GitHub Link

Key Functions

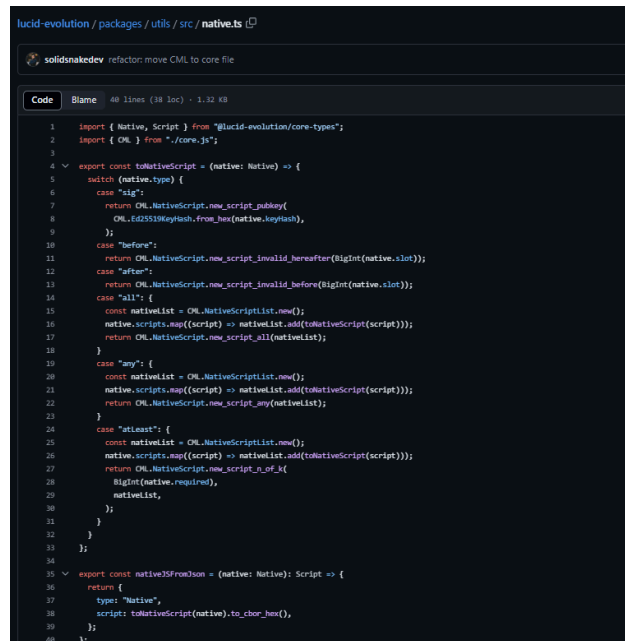
`toNativeScript`

Converts a high-level native script definition into a low-level script that the Cardano node can interpret. This function supports various types of native scripts including simple public key-based scripts, time-lock scripts, and complex multi-script conditions

`nativeJSFromJson`

Encapsulates the conversion of a Native script object into a script that is compatible with the ledger, serialized into CBOR hex format

Code Snapshot



```

lucid-evolution / packages / utils / src / native.ts
solidsnakev refactor: move CML to core file

Code Blame 40 Lines (38 loc) · 1.32 KB

1 import { Native, Script } from "lucid-evolution/core-types";
2 import { OK } from "../core-js";
3
4 export const toNativeScript = (native: Native) => {
5   switch (native.type) {
6     case "sig":
7       return OK.NativeScript.new_script_pubkey(
8         OK.Ed25519KeyHash.from_hex(native.keyhash),
9       );
10    case "before":
11      return OK.NativeScript.new_script_invalid_hereafter(BigInt(native.slot));
12    case "after":
13      return OK.NativeScript.new_script_invalid_before(BigInt(native.slot));
14    case "all": {
15      const nativelist = OK.NativeScriptList.new();
16      native.scripts.map((script) => nativelist.add(toNativeScript(script)));
17      return OK.NativeScript.new_script_all(nativelist);
18    }
19    case "any": {
20      const nativelist = OK.NativeScriptList.new();
21      native.scripts.map((script) => nativelist.add(toNativeScript(script)));
22      return OK.NativeScript.new_script_any(nativelist);
23    }
24    case "atleast": {
25      const nativelist = OK.NativeScriptList.new();
26      native.scripts.map((script) => nativelist.add(toNativeScript(script)));
27      return OK.NativeScript.new_script_n_of_n(
28        BigInt(native.required),
29        nativelist,
30      );
31    }
32  }
33 };
34
35 export const nativeJSFromJson = (native: Native): Script => {
36   return {
37     type: "Native",
38     script: toNativeScript(native).to_cbor_hex(),
39   };
40 };

```

Figure 8: Snapshot-08-Native

Description

The `network.ts` module is to map high-level network identifiers to their corresponding numeric identifiers

Utility Package	Directory
network.ts	GitHub Link

Key Functions

networkToId

Converts a network name into its corresponding numeric ID

Mapping process

```
1  export function networkToId(network: Network): number {
2    switch (network) {
3      case "Preview":
4        return 0;
5      case "Preprod":
6        return 0;
7      case "Custom":
8        return 0;
9      case "Mainnet":
10       return 1;
11     default:
12       throw new Error("Network not found");
13   }
14 }
```

This function's purpose is to ensure that transactions are correctly associated with the appropriate network

Description

The `scripts.ts` module offers a range of functions to manage and transform scripts used in smart contracts. It handles various script types including native, Plutus V1, and Plutus V2 scripts, facilitating their usage in transactions and smart contracts

Key Functions

validatorToAddress

Converts a validator script into a Cardano address

validatorToScriptHash

Generates a script hash from a validator object. This function supports multiple script types including Native, Plutus V1, and Plutus V2

toScriptRef / fromScriptRef

Converts a script into a `CML.Script` object and vice versa, facilitating the use of scripts in a format suitable for transactions

mintingPolicyToId

Converts a minting policy into a policy ID using the script hash functionality

nativeFromJson / nativeScriptFromJson

Converts JSON representations of native scripts into script objects, so that scripts can be handled in a standardized format across the system

applyParamsToScript

Applies parameters to a Plutus script

Utility Package	Directory
scripts.ts	GitHub Link

Code Snapshot

```
lucid-evolution / packages / utils / src / scripts.ts
Code  Blame  157 lines (149 loc) - 4.16 KB

56 export function validatorToScriptHash(validator: Validator): ScriptHash {
57   switch (validator.type) {
58     case "Native":
59       return ONL.NativeScript.from_cbor_hex(validator.script).hash().to_hex();
60     case "PlutusV1":
61       return ONL.PlutusScript.from_v1(
62         ONL.PlutusV1Script.from_cbor_hex(
63           applyDoubleCharEncoding(validator.script),
64         ),
65       ).hash().to_hex();
66     case "PlutusV2":
67       return ONL.PlutusScript.from_v2(
68         ONL.PlutusV2Script.from_cbor_hex(
69           applyDoubleCharEncoding(validator.script),
70         ),
71       ).hash().to_hex();
72     default:
73       throw new Error("No variant matched");
74   }
75 }
76
77 export function toScriptRef(script: Script): ONL.Script {
78   switch (script.type) {
79     case "Native":
80       return ONL.Script.new_native(
81         ONL.NativeScript.from_cbor_hex(script.script),
82       );
83     case "PlutusV1":
84       return ONL.Script.new_plutus_v1(
85         ONL.PlutusV1Script.from_cbor_hex(
86           applyDoubleCharEncoding(script.script),
87         ),
88       );
89     case "PlutusV2":
90       return ONL.Script.new_plutus_v2(
91         ONL.PlutusV2Script.from_cbor_hex(
92           applyDoubleCharEncoding(script.script),
93         ),
94       );
95     default:
96       throw new Error("No variant matched.");
97   }
98 }
99
100 export function fromScriptRef(scriptRef: ONL.Script): Script {
101   const kind = scriptRef.kind();
102   switch (kind) {
103     case "Native":
104       return ONL.NativeScript.from_cbor_hex(scriptRef.script());
105     case "PlutusV1":
106       return ONL.PlutusV1Script.from_cbor_hex(
107         applyDoubleCharEncoding(scriptRef.script()),
108       );
109     case "PlutusV2":
110       return ONL.PlutusV2Script.from_cbor_hex(
111         applyDoubleCharEncoding(scriptRef.script()),
112       );
113     default:
114       throw new Error("No variant matched.");
115   }
116 }
117 }
```

Figure 9: Snapshot-09-Scripts

Description

The `time.ts` module in our library handles the conversion between blockchain-specific slot numbers and Unix timestamps. This functionality is important for scheduling and timing events within the blockchain, where time is often expressed in terms of slots

Utility Package	Directory
time.ts	GitHub Link

Key Functions

unixTimeToSlot

Converts a Unix timestamp to the corresponding slot number in the blockchain. It is to determine when specific events or transactions should occur relative to blockchain time

slotToUnixTime

Converts a slot number to the corresponding Unix timestamp. This allows applications to interpret blockchain time in terms of real-world time

What are slots and how do they serve a role in time?

These functions use `SLOT_CONFIG_NETWORK`, a predefined mapping specific to each network configuration that defines the relationship between slot numbers and Unix time. This ensures accurate time calculations across different network settings

```
1 export function slotToUnixTime(network: Network, slot: Slot): UnixTime {  
2   return slotToBeginUnixTime(slot, SLOT_CONFIG_NETWORK[network]);  
3 }
```

Description

The `utxo.ts` module provides functionality for managing UTXOs. It supports creating transaction inputs and outputs, converting UTXOs to different formats, and sorting or selecting UTXOs based on specific criteria

Utility Package	Directory
utxo.ts	GitHub Link

Key Functions

`utxoToTransactionOutput` / `utxoToTransactionInput`

These functions convert UTXO data into transaction outputs and inputs, facilitating the integration of UTXOs into new transactions

`utxoToCore` / `utxosToCores`

Converts UTXOs to `CML.TransactionUnspentOutput` objects, standardizing UTXOs for transaction processing

`coreToUtxo` / `coresToUtxos`

Reverses the conversion process, transforming `CML.TransactionUnspentOutput` objects back into UTXO format

`selectUTxOs`

Selects UTXOs from a list that meet specified asset requirements, useful in transaction construction where specific asset amounts are required

`sortUTxOs`

Sorts an array of UTXOs according to a specified order, either largest first or smallest first, based on the amount of Lovelace

Code Snapshot

```

lucid-evolution / packages / utx / src / utxo.ts
Code Blame 238 lines (196 loc) · 6.3 KB
55 export function utxosToCores(utxos: Utxo[]): CML.TransactionUnspentOutput[] {
61 }
62
63 // TODO: test coreToUtxo -> utxoToCore strict equality
64 export function coreToUtxo(coreUtxo: CML.TransactionUnspentOutput): Utxo {
65   const out = CML.TransactionOutput.from_chor_hex(coreUtxo.to_chor_hex());
66   const utxo = {
67     ...coreToUtxoOut(coreUtxo),
68     ...coreToUtxoIn(coreUtxo),
69   };
70   return utxo;
71 }
72
73 export function coresToUtxos(utxos: CML.TransactionUnspentOutput[]): Utxo[] {
74   const result: Utxo[] = [];
75   for (let i = 0; i < utxos.length; i++) {
76     result.push(coreToUtxo(utxos[i]));
77   }
78   return result;
79 }
80
81 export function coreToUtxoIn(input: CML.TransactionInput): OutRef {
82   return {
83     txHash: input.transaction_id().to_hex(),
84     outputIndex: parseInt(input.index().toString()),
85   };
86 }
87
88 export function coresToUtxoIn(inputs: CML.TransactionInput[]): OutRef[] {
89   const result: OutRef[] = [];
90   for (let i = 0; i < inputs.length; i++) {
91     result.push(coreToUtxoIn(inputs[i]));
92   }
93   return result;
94 }
95
96 export function coreToUtxoOut(output: CML.TransactionOutput): TxOutput {
97   return {
98     assets: valueToAssets(output.amount()),
99     address: output.address().to_bech32(undefined),
100     datumHash: output.datum()?.as_hash()?.to_hex(),
101     datum: output.datum()?.as_datum()?.to_chor_hex(),
102     scriptRef: output.script_ref() && fromScriptRef(output.script_ref()),
103   };
104 }
105
106 export function coresToUtxoOut(outputs: CML.TransactionOutput[]): TxOutput[] {
107   let result: TxOutput[] = [];
108   for (let i = 0; i < outputs.length; i++) {
109     result.push(coreToUtxoOut(outputs[i]));
110   }
111   return result;
112 }

```

Figure 10: Snapshot-10-Utxo

Description

The `value.ts` module provides functions to manipulate and convert between the blockchain's internal value representation and a more accessible assets format. This serves the purpose for managing transaction outputs and state transitions in smart contracts

Key Functions

`valueToAssets`

Converts a `CML.Value` object, which represents the amount of different tokens in a transaction output, into an `Assets` object that is easier to manipulate and display

`assetsToValue`

Converts an `Assets` object back into a `CML-Value` object for use in transaction creation or other on-chain activities

`fromUnit / toUnit`

These functions handle conversion between a unit representation (combining policy ID and asset names) and its constituent parts, helping in asset identification and manipulation

`addAssets`

Aggregates multiple `Assets` objects into a single object, summing up quantities of the same assets

Utility Package	Directory
value.ts	GitHub Link

Code Snapshot

```

lucid-evolution / packages / utils / src / value.ts
Code Blame 181 lines (96 loc) · 3.21 KB
15  const assetNames = policyAssets.keys();
16  for (let k = 0; k < assetNames.length; k++) {
17    const policyAsset = assetNames.get(k);
18    const quantity = policyAssets.get(policyAsset);
19    //FIXME: report to discard policyAsset.to_chor_hex() adds the head byte twice eg. MyMintToken -> (to Hex) -> 446d79446d
20    const unit = policy.to_hex() + frontText(policyAsset.to_str());
21    assets[unit] = quantity;
22  }
23  }
24  }
25  return assets;
26  }
27
28  export function assetsToValue(assets: Assets): CML.Value {
29    const multiAsset = CML.MultiAsset.new();
30    const lowLace = assets["lowLace"] ? assets["lowLace"] : 0n;
31    const units = Object.keys(assets);
32    const policies = Array.from(
33      new Set(
34        units
35          .filter((unit) => unit !== "lowLace")
36          .map((unit) => unit.slice(0, 56)),
37      ),
38    );
39    for (const policy of policies) {
40      const policyUnits = units.filter((unit) => unit.slice(0, 56) === policy);
41      const assetValue = CML.AssetValue.new();
42      for (const unit of policyUnits) {
43        assetValue.insert(
44          CML.AssetName.from_bytes(frontText(unit.slice(56))),
45          BigInt(assets[unit]),
46        );
47      }
48      multiAsset.insert_asset(CML.ScriptHash.from_hex(policy), assetValue);
49    }
50    return CML.Value.new(lowLace, multiAsset);
51  }
52
53  /**
54   * Splits unit into policy id, asset name (entire asset name), name (asset name without label) and label if applicable.
55   * name will be returned in Hex.
56   */
57  export function fromUnit(unit: Unit): {
58    policyId: string;
59    assetName: string | null;
60    name: string | null;
61    label: number | null;
62  } {
63    const policyId = unit.slice(0, 56);
64    const assetName = unit.slice(56) || null;
65    const label = fromLabel(unit.slice(56, 64));
66    const name = (() => {

```

Figure 11: Snapshot-11-Value

Use Case Scenario

Here's how the Lucid Evolution enabled input indexing would look like, making Staking Validator Design Pattern usage a breeze

```
1  withdraw (
2    rewardAddress: RewardAddress,
3    amount: Lovelace,
4    redeemer?: string | RedeemerBuilder,
5  ) => TxBuilder;
6
7  // The type which needs to be provided in case you want your redeemer to
8  // have input indices but would like lucid to populate them for you
9  // after doing the coin selection
10 export type RedeemerBuilder = {
11   makeRedeemer: (inputIndices: bigint[]) => Redeemer;
12   inputs: UTx0[];
13 };
14
15 const rdmrBuilder: RedeemerBuilder = {
16   makeRedeemer: (inputIndices: bigint[]) => {
17     return Data.to({
18       nodeIdxs: inputIndices,
19       nodeOutIdxs: outputIndices, // you would have this already
20     })),
21   inputs: selectedUTxOs // any inputs that you wish to be indexed, the inputs
22 };
23
24 const tx = lucid_evolution.
25   .newTx()
26   .collectFrom(selectUTxOs, redeemer)
27   .withdraw(rewardAddress, 0n, rdmrBuilder)
28   .attach.SpendingValidator(spend)
29   .attach.WithdrawalValidator(stake)
30   .completeProgram();
```

Testing Suite

Our testing suite, integrated into the Lucid Evolution library through GitHub Actions, automatically runs on push to the main branch and during `pull_request` events. It includes tests in order to ensure each function performs as expected.

Automated tests are triggered to validate code functionality.

Packages

Lucid

- [coinselection.test.ts](#)
- [onchain.test.ts](#)
- [read.test.ts](#)
- [tx.test.ts](#)
- [txHash.test.ts](#)
- [wallet.test.ts](#)

Utils

- [apply-param.test.ts](#)
- [cbor.test.ts](#)
- [native.test.ts](#)
- [utxo.test.ts](#)

Provider

- [koios.test.ts](#)

GIF Testrun

Test Result

Inn this GIF, **an automated test** running in terminal showcases that the above-displayed test packages are passing