ANASTASIA LABS

Proof of Achievement - Milestone 1

Detailed Standing PR report

Project Number 1100024
Project Manager Jonathan Rodriguez

ANASTASIA LABS

Contents

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

Pull Requests (PRs) extracted from legacy Lucid library

Add graphql provider

PR Nr.	Date	Title	Submitter	Link
111	Oct 29, 2022	Add graphql provider	zachykling	GitHub PR

Commits

1. "add graphql as provider" - Initial commit introducing GraphQL as a data provider.
2. "fix datums data for utxos" - Fixes related to the data handling for unspent transaction outputs.

Purpose

Introduces the ability to use a Cardano GraphQL instance as a data provider alongside the Cardano submit API, facilitating the submission of transactions. It aims to leverage community resources like dandelion and freeloaderz.io for enhanced accessibility and efficiency.

450 src/provider/graphql.ts

```
+ import { C, fromHex } from "../mod.ts";
+ import {
+   Address,
+   Assets,
+   CostModels,
+   Delegation,
+   OutRef,
+   ProtocolParameters,
+   Provider,
+   Slot,
+   Transaction,
+   TxHash,
+   Unit,
+   UTxO,
+ } from "../types/mod.ts";
+
+ export class GraphQL implements Provider {
+   gqlUrl: string;
+   submitUrl: string;
+   authToken: string | undefined;
+   constructor(gqlUrl: string, submitUrl: string, authToken?: string) {
+     this.gqlUrl = gqlUrl;
+     this.submitUrl = submitUrl;
+     if (authToken) this.authToken = authToken;
+   }
+
+   async getProtocolParameters(): Promise<ProtocolParameters> {
+     const ProtocolParametersQuery = `
+     query getProtocolParameters {
+       cardano {
+         tip {
+           slotNo
+         }
+         currentEpoch {
+           protocolParams {
+             minFeeA
+             minFeeB
+             poolDeposit
+             keyDeposit
+             coinsPerUtxoByte
```

```
+            maxValSize
+            maxTxSize
+            priceMem
+            priceStep
+            maxTxExMem
+            maxTxExSteps
+            collateralPercent
+            maxCollateralInputs
+            costModels
+          }
+        }
+      }
+    }`;
+    const fullGraphqlQuery = {
+      "operationName": "getProtocolParameters",
+      "query": ProtocolParametersQuery,
+      "variables": {},
+    };
+    const qdata = await this.queryGraphQL(fullGraphqlQuery);
+    const params: ProtocolParamsGQL =
+      qdata.data.cardano.currentEpoch.protocolParams;
+    return {
+      minFeeA: parseInt(params.minFeeA.toString()),
+      minFeeB: parseInt(params.minFeeB.toString()),
+      maxTxSize: parseInt(params.maxTxSize.toString()),
+      maxValSize: parseInt(params.maxValSize.toString()),
+      keyDeposit: BigInt(params.keyDeposit),
+      poolDeposit: BigInt(params.poolDeposit),
+      priceMem: parseFloat(params.priceMem.toString()),
+      priceStep: parseFloat(params.priceStep.toString()),
+      coinsPerUtxoByte: BigInt(params.coinsPerUtxoByte),
+      maxTxExMem: BigInt(params.maxTxExMem),
+      maxTxExSteps: BigInt(params.maxTxExSteps),
+      collateralPercentage: parseInt(params.collateralPercent.toString()),
+      maxCollateralInputs: parseInt(params.maxCollateralInputs.toString()),
+      costModels: params.costModels,
+    };
+  }
+
+  async getCurrentSlot(): Promise<Slot> {
```

```
+     const TipQuery = `
+   query getCurrentTip {
+       cardano {
+         tip {
+           slotNo
+         }
+       }
+     }`;
+     const fullGraphqlQuery = {
+       "operationName": "getCurrentTip",
+       "query": TipQuery,
+       "variables": {},
+     };
+
+     const qdata = await this.queryGraphQL(fullGraphqlQuery);
+
+     const slotNo = qdata.data.cardano.tip.slotNo;
+     if (!slotNo) throw qdata.error;
+
+     return slotNo;
+   }
+
+   async getUtxos(address: string): Promise<UTxO[]> {
+     const UTxOsQuery = `
+     query UTxOsByAddress($address: String!) {
+       utxos(where: { address: { _eq: $address } }) {
+         txHash
+         index
+         value
+         datum {
+           hash
+           bytes
+         }
+         tokens {
+           asset {
+             policyId
+             assetName
+           }
+           quantity
+         }
+       }
```

```
+     }`;
+     const fullGraphqlQuery = {
+       "operationName": "UTxOsByAddress",
+       "query": UTxOsQuery,
+       "variables": { address: address },
+     };
+     const qdata = await this.queryGraphQL(fullGraphqlQuery);
+
+     const utxos: UtxosGraphql = qdata.data?.utxos;
+
+     return utxos.map((r) => ({
+       txHash: r.txHash,
+       outputIndex: r.index,
+       assets: (() => {
+         const a: Assets = {};
+         r.tokens.forEach((token: {
+           asset: {
+             policyId: string;
+             assetName: string;
+           };
+           quantity: string;
+         }) => {
+           a[token.asset.policyId + token.asset.assetName] = BigInt(
+             token.quantity,
+           );
+         });
+         a["lovelace"] = BigInt(r.value);
+         return a;
+       })(),
+       address,
+       datumHash: "",
+     }));
+   }
+
+   async getUtxosWithUnit(address: Address, unit: Unit): Promise<UTxO[]> {
+     const AssetUTxOQuery = `
+   query UTxOWithAssetQuery($address: String!, $policyId: String!, $asset: String!) {
+     utxos(where: {
+       address: { _eq: $address }, _and: {
+       tokens: {
```

```
+          asset: {
+              policyId: {
+                  _eq: $policyId
+                },
+              _and:       {
+                assetName: {
+                  _eq: $asset
+                }
+              }
+            }
+          }
+        }
+      }) {
+        txHash
+        index
+        value
+        datum {
+          hash
+          bytes
+        }
+        transactionOutput {
+          address
+        }
+        tokens {
+          asset {
+            policyId
+            assetName
+          }
+          quantity
+        }
+      }
+    }`;
+
+      const fullGraphqlQuery = {
+        "operationName": "UTxOWithAssetQuery",
+        "query": AssetUTxOQuery,
+        "variables": {
+          address: address,
+          policyId: unit.slice(0, 56),
+          asset: unit.slice(56),
+        },
+      };
```

```
+     };
+     const asstq = await this.queryGraphQL(fullGraphqlQuery);
+
+     const utxos: UtxosGraphql = asstq.data?.utxos;
+     return utxos.map((r) => ({
+       txHash: r.txHash,
+       outputIndex: r.index,
+       assets: (() => {
+         const a: Assets = {};
+         r.tokens.forEach((token: {
+           asset: {
+             policyId: string;
+             assetName: string;
+           };
+           quantity: string;
+         }) => {
+           a[token.asset.policyId + token.asset.assetName] = BigInt(
+             token.quantity,
+           );
+         });
+         a["lovelace"] = BigInt(r.value);
+         return a;
+       })(),
+       address,
+       datumHash: "",
+     }));
+   }
+
+   async awaitTx(txHash: TxHash): Promise<boolean> {
+     const TxQuery = `
+       query TxQuery($txhash: Hash32Hex!) {
+         transactions(where: { hash: { _eq: $txhash } }) {
+           hash
+         }
+       }`;
+     return await new Promise((res, _) => {
+       const confirmation = setInterval(async () => {
+         const fullGraphqlQuery = {
+           "operationName": "TxQuery",
+           "query": TxQuery,
+           "variables": { txhash: txHash },
```

```
+          "variables": { txhash: txHash },
+        };
+        const txQ = await this.queryGraphQL(fullGraphqlQuery);
+
+        if (
+          !txQ.error && !txQ.errors && !txQ.data.transactions &&
+          txQ.data.transactions.length > 0
+        ) {
+          clearInterval(confirmation);
+          res(true);
+          return;
+        }
+      }, 3000);
+    });
+  }
+
+  async submitTx(tx: Transaction): Promise<TxHash> {
+    const transaction = C.Transaction.from_bytes(fromHex(tx));
+    const txhash = C.hash_transaction(transaction.body()).to_hex();
+    const res = await fetch(this.submitUrl, {
+      method: "POST",
+      headers: { "Content-Type": "application/cbor" },
+      body: transaction.to_bytes(),
+    });
+    if (res.status === 200) {
+      return txhash;
+    } else throw res;
+  }
+
+  async getUtxosByOutRef(outRefs: OutRef[]): Promise<UTxO[]> {
+    const q = `query getUtxosByOutRef($outRef: [Hash32Hex]) {
+      utxos(where: {
+        transaction: {
+          hash: {
+            _in: $outRef
+          }
+        }
+      }) {
+        txHash
+        index
+        value
```

```
+          value
+          datum {
+            hash
+            bytes
+          }
+          tokens {
+            asset {
+              policyId
+              assetName
+            }
+            quantity
+          }
+          transactionOutput {
+            address
+          }
+        }
+      }`;
+
+      const queryHashes = [...new Set(outRefs.map((outRef) => outRef.txHash))];
+      const fullGraphqlQuery = {
+        "operationName": "getUtxosByOutRef",
+        "query": q,
+        "variables": { outRef: queryHashes },
+      };
+      const utxos: UtxosGraphql = await this.queryGraphQL(fullGraphqlQuery);
+
+      return graphqlSchemaUtxosToUtxos(
+        utxos.reduce((acc: UtxosGraphql, utxos) => acc.concat(utxos), []).filter((
+          utxo,
+        ) =>
+          outRefs.some((outRef) =>
+            utxo.txHash === outRef.txHash && utxo.index === outRef.outputIndex
+          )
+        ),
+      );
+    }
+    async getDelegation(rewardAddress: string): Promise<Delegation> {
+      const q = `
+      query getDelegation($address: String!){
+        rewards(where: { address: {_eq: $address}}) {
+          amount
```

```
+          amount
+          stakePool {
+            id
+          }
+        }
+      }`;
+      const fullGraphqlQuery = {
+        "operationName": "getDelegation",
+        "query": q,
+        "variables": { address: rewardAddress },
+      };
+      const dQ = await this.queryGraphQL(fullGraphqlQuery);
+      if (dQ.data && dQ.data.length > 0) {
+        return { rewards: dQ.data[0].amount, poolId: dQ.data[0].stakePool.id };
+      } else if (dQ.error) throw dQ.error;
+      return { rewards: 0n, poolId: null };
+    }
+
+    async getDatum(datumHash: string) {
+      //currently it's not possible to filter out records where datum doesn't have bytes
+      //TODO: watch out for future releases of Cardano graphql
+      const q = `query getDatumFromHash($datumHash: Hash32Hex!) {
+        utxos(where: {
+          datum: {hash : { _eq: $datumHash}}
+        }) {
+            datum {
+              bytes
+            }
+        }
+      }
+      `;
+      const fullGraphqlQuery = {
+        "operationName": "getDatumFromHash",
+        "query": q,
+        "variables": { datumHash: datumHash },
+      };
+      const dQ = await this.queryGraphQL(fullGraphqlQuery);
+      if (dQ.data) {
+        for (const r of dQ.data) {
+          if (r.datum.bytes) return r.datum.bytes;
+        }
```

```
+       }
+     } else if (dQ.error) throw dQ.error;
+     return null;
+   }
+
+   async queryGraphQL(fullGraphqlQuery: {
+     operationName: string;
+     query: string;
+     // deno-lint-ignore no-explicit-any
+     variables: any;
+     // deno-lint-ignore no-explicit-any
+   }): Promise<any> {
+     const headers: {
+       "content-type": string;
+       "Authorization"?: string;
+     } = {
+       "content-type": "application/json",
+     };
+
+     if (this.authToken) headers.Authorization = `Bearer ${this.authToken}`;
+
+     const options = {
+       "method": "POST",
+       "headers": headers,
+       "body": JSON.stringify(fullGraphqlQuery),
+     };
+     const response = await fetch(this.gqlUrl, options);
+     return await response.json();
+   }
+ }
+
+ type ProtocolParamsGQL = {
+   minFeeA: number;
+   minFeeB: number;
+   poolDeposit: number;
+   keyDeposit: number;
+   coinsPerUtxoByte: number;
+   maxValSize: string;
+   maxTxSize: number;
+   priceMem: number;
+   priceStep: number;
```

```
+   maxTxExMem: string;
+   maxTxExSteps: string;
+   collateralPercent: number;
+   maxCollateralInputs: number;
+   costModels: CostModels;
+ };
+
+ type UtxosGraphql = {
+   txHash: string;
+   index: number;
+   value: string;
+   tokens: {
+     asset: {
+       policyId: string;
+       assetName: string;
+     };
+     quantity: string;
+   }[];
+   transactionOutput: {
+     address: string;
+   };
+ }[];
+
+ function graphqlSchemaUtxosToUtxos(utxos: UtxosGraphql): UTxO[] {
+   return utxos.map((r) => ({
+     txHash: r.txHash,
+     outputIndex: r.index,
+     assets: (() => {
+       const a: Assets = {};
+       r.tokens.forEach((token: {
+         asset: {
+           policyId: string;
+           assetName: string;
+         };
+         quantity: string;
+       }) => {
+         a[token.asset.policyId + token.asset.assetName] = BigInt(
+           token.quantity,
+         );
+       });
```

Fix typebox portable type annotation

PR Nr.	Date	Title	Submitter	Link
197	Jun 7, 2023	Fix typebox portable type annotation	solidsnakedev Link	GitHub PR

Commits

1. fix typebox portable type annonation

Purpose

This PR introduces a critical fix for type annotation in TypeScript configurations. The main objective is to resolve an issue where the TypeScript compiler could not name the inferred type of 'CredentialSchema' without a reference to a specific module in node_modules. This problem impacts portability and can hinder the deployment of the project across different environments.

Technical Changes

```
export const CredentialSchema = Data.Enum([
  Data.Object({
    PublicKeyCredential: Data.Tuple([
      Data.Bytes({ minLength: 28, maxLength: 28 }),
    ]),
  }),
  Data.Object({
    ScriptCredential: Data.Tuple([
      Data.Bytes({ minLength: 28, maxLength: 28 }),
    ]),
  }),
]);
export type Credential = Data.Static<typeof CredentialSchema>;
export const Credential = CredentialSchema as unknown as
Credential;
```

Issue addressed

The patch fixes an issue with non-portable type annotations that arise due to the inferred type references, which are essential for project deployments that require explicit type declarations as part of their TypeScript configuration.

Wallet from seed should query both base and enterprise addresses on signing

PR Nr.	Date	Title	Submitter	Link
217	Aug 28, 2023	Wallet from seed should query both base and enterprise addresses on signing	infrmtcs	GitHub PR

Commits

1. Wallet from seed should query both base and enterprise addresses on signing

Purpose

PR 217 addresses a critical issue in the transaction signing process when using wallets derived from seeds. The problem stems from the library's method of querying UTXOs. Currently, the library only queries by base address, ignoring the payment credential. This results in the library refusing to sign transactions even when the user holds valid signing authority, leading to transaction failures.

Technical Changes

```
const utxos = await this.utxosAt(address);
```

This line of code is crucial as it is responsible for fetching UTXO data which is necessary for signing transactions. The issue was identified during the signTx call, where the function failed to correctly query UTXOs based on the payment credential.

Issue addressed

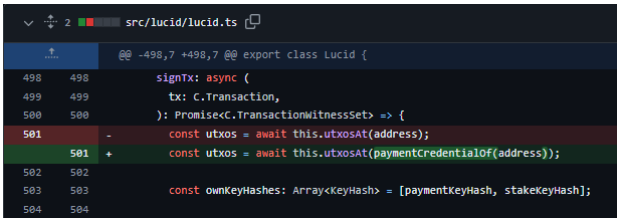
The specific bug involves the incorrect assumption that the addressType is unset, which leads to querying by base address rather than by the more specific payment credential. This misalignment causes valid signers to be unrecognized by the system during transaction signing, as showcased in the reproduction case provided.

Reproduction Case and Error

Transaction Details (from cardano-cli on preview testnet):

Inputs, outputs, and other transaction details reflecting the specific addresses and amounts affected.

Error Thrown: ShelleyTxValidationError related to missing VKey witnesses, indicating a failure in the authentication of transactions due to the incorrect UTXO query logic.



Prototype Hydra Provider

PR Nr.	Date	Title	Submitter	Link
218	Aug 29, 2023	Prototype Hydra provider	Piefayth	GitHub PR

Commits

1. **Initial Commit (1f423e4):** Initial draft of Hydra provider
2. **Subsequent Update (767777b):** Minor bugfix and cleanup in Hydra provider

Purpose

This PR introduces a prototype for a Hydra provider, which is aimed at enhancing interactions with the Hydra node directly from the browser. The prototype addresses issues related to Cross-Origin Resource Sharing (CORS) and the management of UTXO sets within a browser environment.

Technical Changes

The provider is designed to retrieve the entire UTXO set from the Hydra node, which could be a significant change depending on the application's requirements and the size of the data involved. It incorporates a solution for enabling CORS when calling from the browser, which is crucial for web-based applications interfacing with blockchain networks.

300 src/provider/hydra.ts

```
+ import { Provider } from "../types/mod.ts";
+ import {
+   Credential,
+   Delegation,
+   OutRef,
+   ProtocolParameters,
+   UTxO,
+ } from "../types/types.ts";
+ import { Assets } from "../types/mod.ts";
+ import { C } from "../core/mod.ts";
+ import { fromHex, getAddressDetails } from "../mod.ts";
+
+ type HydraCommand =
+   | { tag: "GetUTxO" }
+   | { tag: "NewTx"; transaction: string };
+
+ interface ServerResponse {
+   tag: string;
+   timestamp: Date;
+   seq: number;
+   headId: string;
+ }
+
+ interface GetUTxOResponse extends ServerResponse {
+   tag: "GetUTxOResponse";
+   utxo: Utxos;
+ }
+
+ interface TxValid extends ServerResponse {
+   tag: "TxValid";
+   transaction: Transaction;
+ }
+
+ interface TxInvalid extends ServerResponse {
+   tag: "TxInvalid";
+   transaction: Transaction;
+   validationError: {
+     reason: string;
+   };
+ }
+
```

```
+ interface Transaction {
+   id: string;
+ }
+
+ type Utxo = {
+   address: string;
+   datumhash?: string | undefined;
+   inlineDatum?: string | undefined;
+   referenceScript?: string | undefined;
+   value: Assets;
+ };
+
+ type Utxos = {
+   [key: string]: Utxo;
+ };
+
+ export class Hydra implements Provider {
+   wsUrl: string;
+   httpUrl: string;
+
+   constructor(
+     host: string,
+     ssl: boolean = false,
+   ) {
+     this.wsUrl = `${ssl ? "wss" : "ws"}://${host}?history=no&snapshot-utxo=no`;
+     this.httpUrl = `${ssl ? "https" : "http"}://${host}`;
+   }
+
+   async getProtocolParameters(): Promise<ProtocolParameters> {
+     const result = await (
+       await fetch(`${this.httpUrl}/protocol-parameters`)
+     ).json();
+
+     return {
+       minFeeA: parseInt(result.txFeePerByte),
+       minFeeB: parseInt(result.txFeeFixed),
+       maxTxSize: parseInt(result.maxTxSize),
+       maxValSize: parseInt(result.maxValueSize),
+       keyDeposit: BigInt(result.stakeAddressDeposit),
+       poolDeposit: BigInt(result.stakePoolDeposit),
+       priceMem: parseFloat(result.executionUnitPrices.priceMemory),
```

```
+       priceStep: parseFloat(result.executionUnitPrices.priceSteps),
+       maxTxExMem: BigInt(result.maxTxExecutionUnits.memory),
+       maxTxExSteps: BigInt(result.maxTxExecutionUnits.steps),
+       coinsPerUtxoByte: BigInt(result.txFeePerByte),
+       collateralPercentage: parseInt(result.collateralPercentage),
+       maxCollateralInputs: parseInt(result.maxCollateralInputs),
+       costModels: {
+         "PlutusV1": result.costModels.PlutusV1 || {},
+         "PlutusV2": result.costModels.PlutusV2 || {},
+       },
+     };
+   }
+
+   async getUtxos(addressOrCredential: string | Credential): Promise<UTxO[]> {
+     return (await this.getSnapshotUtxos())
+       .filter((utxo) => {
+         if (typeof addressOrCredential === "string") {
+           return addressOrCredential === utxo.address;
+         } else {
+           const { paymentCredential } = getAddressDetails(
+             utxo.address,
+           );
+           paymentCredential?.hash;
+           return paymentCredential?.hash === addressOrCredential.hash;
+         }
+       });
+   }
+
+   async getUtxosWithUnit(
+     addressOrCredential: string | Credential,
+     unit: string,
+   ): Promise<UTxO[]> {
+     const utxos = await this.getUtxos(addressOrCredential);
+     return utxos.filter((utxo) => utxo.assets[unit] > 0n);
+   }
+
+   async getUtxoByUnit(unit: string): Promise<UTxO> {
+     const utxos = (await this.getSnapshotUtxos())
+       .filter((utxo) => utxo.assets[unit] > 0n);
+
+     if (utxos.length > 1) {
```

```
+       throw new Error("Unit needs to be an NFT or only held by one address.");
+     }
+
+     if (utxos.length < 1) {
+       throw new Error("Unit not found at any address.");
+     }
+
+     return utxos[0];
+   }
+
+   async getUtxosByOutRef(outRefs: OutRef[]): Promise<UTxO[]> {
+     const client = await this.hydraWsp({ tag: "GetUTxO" });
+     const utxoResponse = await this.awaitMessage<GetUTxOResponse>(client);
+
+     client.close();
+
+     return outRefs.flatMap((outRef) => {
+       const concatenatedRef = `${outRef.txHash}#${outRef.outputIndex}`;
+       const maybeUtxo = utxoResponse.utxo[concatenatedRef];
+
+       return maybeUtxo ? this.convertHydraUtxo(concatenatedRef, maybeUtxo) : [];
+     });
+   }
+
+   getDelegation(rewardAddress: string): Promise<Delegation> {
+     throw new Error("Delegation does not apply to Hydra.");
+   }
+
+   async getDatum(datumHash: string): Promise<string> {
+     return (await this.getSnapshotUtxos())
+       .filter((utxo) => utxo.datumHash === datumHash)[0].datum!;
+   }
+
+   async awaitTx(
+     txHash: string,
+     checkInterval?: number | undefined,
+   ): Promise<boolean> {
+     const client = new WebSocket(this.wsUrl);
+     await new Promise((res) => {
+       client.addEventListener("open", () => res(1), { once: true });
+     });
```

```
+     const isValid = await this.awaitTxValid(txHash, client, checkInterval);
+     client.close();
+     return isValid;
+   }
+
+   async submitTx(tx: string): Promise<string> {
+     const client = await this.hydraWsp({
+       tag: "NewTx",
+       transaction: tx,
+     });
+
+     client.close();
+
+     const coreTx = C.Transaction.from_bytes(fromHex(tx));
+     const txHash = C.hash_transaction(coreTx.body()).to_hex();
+     return txHash;
+   }
+
+   private async getSnapshotUtxos(): Promise<UTxO[]> {
+     const client = await this.hydraWsp({ tag: "GetUTxO" });
+     const utxoResponse = await this.awaitMessage<GetUTxOResponse>(client);
+
+     client.close();
+
+     return Object.entries(utxoResponse.utxo)
+       .map(([outputRef, utxo]) => {
+         return this.convertHydraUtxo(outputRef, utxo);
+       });
+   }
+
+   private convertHydraUtxo(outputRef: string, utxo: Utxo): UTxO {
+     const [txHash, outputIndex] = outputRef.split("#");
+
+     return {
+       txHash,
+       outputIndex: Number(outputIndex),
+       assets: utxo.value,
+       address: utxo.address,
+       datumHash: utxo.datumhash,
+       datum: utxo.inlineDatum,
+       scriptRef: utxo.referenceScript
```

```
+          ? {
+            type: "PlutusV2",
+            script: utxo.referenceScript,
+          }
+          : undefined,
+      };
+    }
+
+    private async awaitMessage<T>(client: WebSocket): Promise<T> {
+      return await new Promise((res, rej) => {
+        client.addEventListener("message", (msg: MessageEvent<string>) => {
+          try {
+            const serverResponse = JSON.parse(msg.data);
+            if (serverResponse.tag == "CommandFailed") {
+              rej(
+                new Error(
+                  `Received "Command Failed" from Hydra. Is Hydra not in the right state?`,
+                ),
+              );
+            } else {
+              res(serverResponse as T);
+            }
+          } catch (e) {
+            rej(e);
+          }
+        }, { once: true });
+      });
+    }
+
+    private async awaitTxValid(
+      txHash: string,
+      client: WebSocket,
+      timeoutMs: number | undefined = 5000,
+    ): Promise<boolean> {
+      return await new Promise((res, rej) => {
+        const listener = (msg: MessageEvent<string>) => {
+          try {
+            const serverResponse = JSON.parse(msg.data) as ServerResponse;
+            if (serverResponse.tag == "CommandFailed") {
+              rej(
+                new Error(
```

```
+                `Received "Command Failed" from Hydra. Is Hydra not in the right state?`,
+              ),
+            );
+          } else if (serverResponse.tag == "TxValid") {
+            if ((serverResponse as TxValid).transaction.id !== txHash) {
+              return;
+            }
+            client.removeEventListener("message", listener);
+            res(true);
+          } else if (serverResponse.tag == "TxInvalid") {
+            if ((serverResponse as TxInvalid).transaction.id !== txHash) {
+              return;
+            }
+            client.removeEventListener("message", listener);
+            rej(serverResponse);
+          }
+        } catch (e) {
+          client.removeEventListener("message", listener);
+          rej(e);
+        }
+      };
+
+      client.addEventListener("message", listener);
+
+      /* If the user calls awaitTxValid in an inappropriate way, it
+         may leak the client and listeners. This timeout guarantees cleanup. */
+
+      setTimeout(() => {
+        if (client.readyState !== WebSocket.CLOSING || WebSocket.CLOSED) {
+          client.removeEventListener("message", listener);
+          rej(
+            new Error(`Hydra never reported success or failure of ${txHash}`),
+          );
+        }
+      }, timeoutMs);
+    });
+  }
+
+  private async hydraWsp(
+    command: HydraCommand,
+  ): Promise<WebSocket> {
```

Add Koios Provider to Lucid

PR Nr.	Date	Title	Submitter	Link
219	Sep 3, 2023	Add Koios Provider to Lucid	edridudi	GitHub PR

Commits

1. Initial commits set up the basic structure and functionality
2. Follow-up commits incorporated feedback, added documentation, and handled minor bug fixes and linting issues

Purpose

PR 219 introduces the Koios Provider to the Lucid library, enhancing the library's capabilities by integrating with the Koios API. This provider enables the Lucid library to fetch blockchain data through the Koios service, supporting various network environments such as Mainnet, Preview, and Preprod.

Technical Changes

The Koios Provider class implements the Provider interface, handling data retrieval from the Koios API.It includes methods for fetching protocol parameters, UTXOs by address or credential, and specific UTXOs by unit or output reference. The provider handles network-specific configurations and error management, aiming to streamline interactions with the Cardano blockchain through Koios.

```
7   -   "repository": "https://github.com/spacebudz/lucid"
7   +   "repository": "https://github.com/spacebudz/lucid",
8   +   "dependencies": {
9   +     "@adabox/koios-ts-client": "^1.0.6"
10  +   }
```

Figure 20: package.json

236 src/provider/koios.ts

```
+ import {
+     Address,
+     Assets,
+     Credential,
+     Datum,
+     DatumHash,
+     Delegation,
+     Network,
+     OutRef,
+     ProtocolParameters,
+     Provider,
+     RewardAddress,
+     Transaction,
+     TxHash,
+     Unit,
+     UTxO,
+ } from "../types/mod.ts";
+ import {BackendFactory, KoiosHttpError, KoiosTimeoutError} from "@adabox/koios-ts-client/dist/index.js"
+ import {C} from "../core/core.ts";
+ import {applyDoubleCborEncoding, fromHex, fromUnit} from "../utils/utils.ts";
+
+ export class KoiosProvider implements Provider {
+
+     private readonly backendService
+
+     constructor(network: Network) {
+         if (network === 'Mainnet') {
+             this.backendService = BackendFactory.getKoiosMainnetService()
+         } else if (network === 'Preview') {
+             this.backendService = BackendFactory.getKoiosPreviewService()
+         } else if (network === 'Preprod') {
+             this.backendService = BackendFactory.getKoiosPreprodService()
+         } else {
+             throw Error("Unsupported Network Type")
+         }
+     }
+
+     async getProtocolParameters(): Promise<ProtocolParameters> {
+         const result = await this.backendService.getEpochService().getEpochProtocolParameters()
+
+         return {
```

Figure 21: src/provider/koios.ts

```
+            minFeeA: parseInt(result[0].min_fee_a),
+            minFeeB: parseInt(result[0].min_fee_b),
+            maxTxSize: parseInt(result[0].max_tx_size),
+            maxValSize: parseInt(result[0].max_val_size),
+            keyDeposit: BigInt(result[0].key_deposit),
+            poolDeposit: BigInt(result[0].pool_deposit),
+            priceMem: parseFloat(result[0].price_mem),
+            priceStep: parseFloat(result[0].price_step),
+            maxTxExMem: BigInt(result[0].max_tx_ex_mem),
+            maxTxExSteps: BigInt(result[0].max_tx_ex_steps),
+            coinsPerUtxoByte: BigInt(result[0].coins_per_utxo_size),
+            collateralPercentage: parseInt(result[0].collateral_percent),
+            maxCollateralInputs: parseInt(result[0].max_collateral_inputs),
+            costModels: JSON.parse(result[0].cost_models),
+        };
+    }
+
+    async getUtxos(addressOrCredential: Address | Credential): Promise<UTxO[]> {
+        const queryPredicate = (() => {
+            if (typeof addressOrCredential === "string") return addressOrCredential;
+             // should be 'script' (CIP-0005)
+            return addressOrCredential.type === "Key"
+                ? C.Ed25519KeyHash.from_hex(addressOrCredential.hash).to_bech32("addr_vkh")
+                : C.ScriptHash.from_hex(addressOrCredential.hash).to_bech32("addr_vkh");
+        })();
+        try {
+            const result = await this.backendService.getAddressService().getAddressInformation([queryPredicate])
+            if (Array.isArray(result) && result.length > 0 && result[0].utxo_set && result[0].utxo_set.length > 0) {
+                return this.koiosUtxosToUtxos(result[0].utxo_set, result[0].address)
+            } else {
+                return []
+            }
+        } catch (e) {
+            throw new Error("Could not fetch UTxOs from Koios. Try again.");
+        }
+    }
+
+    private async koiosUtxosToUtxos(result: any, address?: string): Promise<UTxO[]> {
+        return (await Promise.all(
+            result.map(async (r: any) => ({
+                txHash: r.tx_hash,
```

```
+                outputIndex: r.tx_index,
+                assets: (() => {
+                    const a: Assets = {};
+                    r.asset_list.forEach((am: any) => {
+                        a[am.policy_id + am.asset_name] = BigInt(am.quantity);
+                    });
+                    return a;
+                })(),
+                address: address ? address : r.payment_addr.bech32,
+                datumHash: !r.inline_datum ? r.datum_hash : undefined,
+                datum: r.inline_datum,
+                scriptRef: {
+                    type: r.reference_script ? r.reference_script.type : null,
+                    script: r.reference_script ? applyDoubleCborEncoding(r.reference_script.bytes) : null
+                },
+            })),
+        )) as UTxO[];
+    }
+
+    async getUtxosWithUnit(addressOrCredential: Address | Credential, unit: Unit): Promise<UTxO[]> {
+        const queryPredicate = (() => {
+            if (typeof addressOrCredential === "string") return addressOrCredential;
+            // should be 'script' (CIP-0005)
+            return addressOrCredential.type === "Key"
+                ? C.Ed25519KeyHash.from_hex(addressOrCredential.hash).to_bech32("addr_vkh")
+                : C.ScriptHash.from_hex(addressOrCredential.hash).to_bech32("addr_vkh");
+        })();
+        try {
+            const result = await this.backendService.getAddressService().getAddressInformation([queryPredicate])
+            if (Array.isArray(result) && result.length > 0 && result[0].utxo_set && result[0].utxo_set.length > 0) {
+                return (await this.koiosUtxosToUtxos(result[0].utxo_set, result[0].address)).filter((utxo): utxo is UTxO => {
+                    const keys = Object.keys(utxo.assets)
+                    return keys.length > 0 && keys.includes(unit)
+                })
+            } else {
+                return []
+            }
+        } catch (e) {
+            throw new Error("Could not fetch UTxOs from Koios. Try again.");
+        }
+    }
```

```
+
+    async getUtxoByUnit(unit: Unit): Promise<UTxO> {
+        let assetAddresses
+        try {
+            let { policyId, assetName } = fromUnit(unit)
+            assetName = String(assetName)
+            assetAddresses = await this.backendService.getAssetService().getAssetAddresses(policyId, assetName)
+        } catch (e) {
+            throw new Error("Could not fetch UTxO from Koios. Try again.");
+        }
+        if (Array.isArray(assetAddresses) && assetAddresses.length > 0) {
+            if (assetAddresses.length > 1) {
+                throw new Error("Unit needs to be an NFT or only held by one address.");
+            }
+            const address = assetAddresses[0].payment_address
+            try {
+                const utxos: UTxO[] = await this.getUtxos(address)
+                const result = utxos.find<UTxO>((utxo): utxo is UTxO => {
+                    const keys = Object.keys(utxo.assets)
+                    return keys.length > 0 && keys.includes(unit)
+                })
+                if (result) {
+                    return result
+                }
+            } catch (e) {
+                throw new Error("Could not fetch UTxO from Koios. Try again.");
+            }
+        }
+        throw new Error("Unit not found.");
+    }
+
+    async getUtxosByOutRef(outRefs: OutRef[]): Promise<UTxO[]> {
+        try {
+            const utxos = []
+            const queryHashes = [...new Set(outRefs.map((outRef) => outRef.txHash))];
+            const result = await this.backendService.getTransactionsService().getTransactionUTxOs(queryHashes)
+            if (Array.isArray(result) && result.length > 0) {
+                for (const utxo of result) {
+                    if (utxo.outputs && utxo.outputs.length > 0) {
+                        utxos.push(await this.koiosUtxosToUtxos(utxo.outputs))
+                    }
```

```
165 +                }
166 +                return utxos.reduce((acc, utxos) => acc.concat(utxos), []).filter((utxo) =>
167 +                    outRefs.some((outRef) =>
168 +                        utxo.txHash === outRef.txHash && utxo.outputIndex === outRef.outputIndex
169 +                    )
170 +                );
171 +            } else {
172 +                return []
173 +            }
174 +        } catch (e) {
175 +            throw new Error("Could not fetch UTxOs from Koios. Try again.");
176 +        }
177 +    }
178 +
179 +    async getDelegation(rewardAddress: RewardAddress): Promise<Delegation> {
180 +        try {
181 +            const result = await this.backendService.getAccountService().getAccountInformation([rewardAddress])
182 +            if (Array.isArray(result) && result.length > 0) {
183 +                return {
184 +                    poolId: result[0].delegated_pool || null,
185 +                    rewards: BigInt(result[0].rewards_available),
186 +                }
187 +            }
188 +        } catch (e) {
189 +            throw new Error("Could not fetch Account Information from Koios. Try again.");
190 +        }
191 +        throw new Error("No Delegation Found by Reward Address");
192 +    }
193 +
194 +    async getDatum(datumHash: DatumHash): Promise<Datum> {
195 +        try {
196 +            const result = await this.backendService.getScriptService().getDatumInformation([datumHash])
197 +            if (Array.isArray(result) && result.length > 0) {
198 +                return result[0].bytes
199 +            }
200 +        } catch (e) {
201 +            throw new Error("Could not fetch Datum Information from Koios. Try again.");
202 +        }
203 +        throw new Error("No Datum Found by Datum Hash");
204 +    }
205 +
```

```
+     awaitTx(txHash: TxHash, checkInterval = 3000): Promise<boolean> {
+         return new Promise((res) => {
+             const confirmation = setInterval(async () => {
+                 try {
+                     const result = await this.backendService.getTransactionsService().getTransactionInformation([txHash])
+                     if (Array.isArray(result) && result.length > 0) {
+                         clearInterval(confirmation);
+                         await new Promise((res) => setTimeout(() => res(1), 1000));
+                         return res(true)
+                     }
+                 } catch (e) {
+                     throw new Error("Could not fetch Datum Information from Koios. Try again.");
+                 }
+             }, checkInterval);
+         });
+     }
+
+     async submitTx(tx: Transaction): Promise<TxHash> {
+         try {
+             return await this.backendService.getTransactionsService().submitTransaction(fromHex(tx))
+         } catch (e) {
+             if (e instanceof KoiosHttpError) {
+                 throw new Error(`Transaction Submission Error: ${e.message}`);
+             } else if (e instanceof KoiosTimeoutError) {
+                 throw new Error("Timeout Error.");
+             } else {
+                 throw new Error("Could not submit transaction.");
+             }
+         }
+     }
+ }
```

```diff
src/provider/mod.ts
@@ -1,3 +1,4 @@
 export * from "./blockfrost.ts";
 export * from "./kupmios.ts";
 export * from "./emulator.ts";
+ export * from "./koios.ts";
```

Figure 27: src/provider/mod.ts

95 tests/koios.test.ts

```diff
@@ -0,0 +1,95 @@
+ import {KoiosProvider} from "../src/provider/koios.ts";
+ import {Datum, Delegation, ProtocolParameters, UTxO} from "../src/types/types.ts";
+ import {assert} from "https://deno.land/std@0.145.0/testing/asserts.ts";
+
+ Deno.test("getProtocolParameters", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const pp: ProtocolParameters = await koios.getProtocolParameters();
+         assert(pp);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getUtxos", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const utxos: UTxO[] = await koios.getUtxos("addr1qy2jt0qpqz2z2z9zx5w4xemekkce7yderz53kjue53lpqv90lkfa9sgrfjuz6uvt4uqtrqhl2kj0a9lnr9ndzutx32gqleeckv");
+         assert(utxos);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getUtxosWithUnit", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const utxos: UTxO[] = await koios.getUtxosWithUnit(
+             "addr1q8vaadv0h7atv366u6966u4rft2svjlf5uajy8lkpsgdrc24rnskuetxz2u3m5ac22s3njvftxcl2fc8k8kjr088ge0qpn6xhn",
+             "85152e10643c1440ba2ba817e3dd1faf7bd7296a8b605efd0f0f2d1844696d656e73696f6e426f78202330313739");
+         assert(utxos);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getUtxoByUnit", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const utxo: UTxO = await koios.getUtxoByUnit("85152e10643c1440ba2ba817e3dd1faf7bd7296a8b605efd0f0f2d1844696d656e73696f6e426f78202330313739");
```

```
+         assert(utxo);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getUtxosByOutRef", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const utxos: UTxO[] = await koios.getUtxosByOutRef([{txHash: 'c6ee20549eab1e565a4bed119bb8c7fc2d11cc5ea5e1e25433a34f0175c0bef6', outputIndex: 0}]);
+         assert(utxos);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getDelegation", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const delegation: Delegation = await koios.getDelegation('stake1uyrx65wjqjgeeksd8hptmcgl5jfyrqkfq0xe8xlp367kphsckq250');
+         assert(delegation);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("getDatum", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const datum: Datum = await koios.getDatum('818ee3db3bbbd04f9f2ce21778cac3ac605802a4fcb00c8b3a58ee2dafc17d46');
+         assert(datum);
+     } catch (e) {
+         console.log(e)
+     }
+ });
+
+ Deno.test("awaitTx", async () => {
+     const koios = new KoiosProvider("Mainnet")
+     try {
+         const isConfirmed: boolean = await koios.awaitTx('f144a8264acf4bdfe2e1241170969c930d64ab6b0996a4a45237b623f1dd670e');
+         assert(isConfirmed);
```

Feat (Next.JS): Support serverside Next.JS usage

PR Nr.	Date	Title	Submitter	Link
220	Sep 6, 2023	Feat (Next.JS): Support serverside Next.JS usage	thaddeusdiamond	GitHub PR

Commits

1. Initial commits set up the basic structure and functionality
2. Follow-up commits incorporated feedback, added documentation, and handled minor bug fixes and linting issues

Purpose

PR 220 aims to address compatibility issues with server-side usage of Next.JS, particularly concerning relative imports which have been disallowed since version 12.0.1 of Next.JS. The pull request introduces a method to detect the Next.JS environment and use a fallback URL if necessary

Related Issue

PR 174 which details a bug where properties of undefined are being read due to Next.JS's updated handling of imports

Technical Changes

Detects if the runtime environment is Next.JS and adjusts import paths accordingly to avoid issues with middleware that disallow relative URLs.

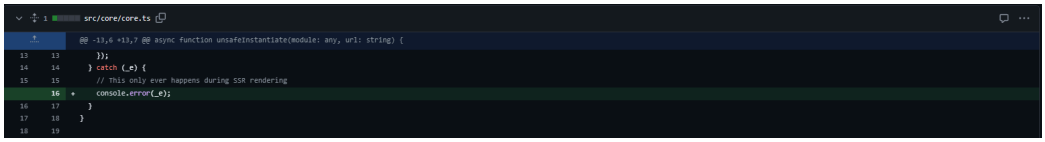


Figure 30: src/core/core.ts

```
15 src/core/libs/cardano_message_signing/cardano_message_signing.generated.js

@@ -3806,12 +3806,25 @@ export function isInstantiated() {
3806  3806    return instanceWithExports != null;
3807  3807  }
3808  3808
      3809 + function requiresFallbackUrl() {
      3810 +   if (import.meta.url.includes("_frsh")) {
      3811 +     return true;
      3812 +   }
      3813 +   if (globalThis.process?.env?.VERCEL_ENV === 'production') {
      3814 +     return true;
      3815 +   }
      3816 +   if (globalThis.process?.env?.NEXT_RUNTIME === 'nodejs') {
      3817 +     return true;
      3818 +   }
      3819 +   return false;
      3820 + }
      3821 +
3809  3822  /**
3810  3823   * @param {InstantiateOptions} opts
3811  3824   */
3812  3825  async function instantiateModule(opts) {
3813  3826    // Temporary exception for fresh framework
3814       - const wasmUrl = import.meta.url.includes("_frsh")
      3827 + const wasmUrl = requiresFallbackUrl()
3815  3828      ? opts.url
3816  3829      : new URL("cardano_message_signing_bg.wasm", import.meta.url);
3817  3830    const decompress = opts.decompress;
```

Figure 31: src/core/core.ts

15 src/core/libs/cardano_multiplatform_lib/cardano_multiplatform_lib.generated.js

```diff
@@ -28591,12 +28591,25 @@ export function isInstantiated() {
28591 28591   return instanceWithExports != null;
28592 28592 }
28593 28593
      28594 + function requiresFallbackUrl() {
      28595 +   if (import.meta.url.includes("_frsh")) {
      28596 +     return true;
      28597 +   }
      28598 +   if (globalThis.process?.env?.VERCEL_ENV === 'production') {
      28599 +     return true;
      28600 +   }
      28601 +   if (globalThis.process?.env?.NEXT_RUNTIME === 'nodejs') {
      28602 +     return true;
      28603 +   }
      28604 +   return false;
      28605 + }
      28606 +
28594 28607 /**
28595 28608  * @param {InstantiateOptions} opts
28596 28609  */
28597 28610 async function instantiateModule(opts) {
28598 28611   // Temporary exception for fresh framework
28599     - const wasmUrl = import.meta.url.includes("_frsh")
      28612 + const wasmUrl = requiresFallbackUrl()
28600 28613     ? opts.url
28601 28614     : new URL("cardano_multiplatform_lib_bg.wasm", import.meta.url);
28602 28615   const decompress = opts.decompress;
```

Figure 32: src/core/core.ts

Feat: chaining of txs

PR Nr.	Date	Title	Submitter	Link
231	Oct 30, 2023	Feat: chaining of txs	will-break-it	GitHub PR

Commits

1. feat: chaining of txs

Purpose

PR 231 introduces a feature to enable chaining of transactions directly via the API. This new functionality allows transactions to be seamlessly linked, where outputs from one transaction can be used as inputs for subsequent transactions.

This enhancement is particularly useful in complex transaction scenarios where multiple, dependent transactions need to be processed in sequence

Technical Changes

The addition of a chain method to the transaction API which facilitates the selection and use of transaction outputs as inputs for another transaction. The inclusion of a test case to demonstrate and verify the chaining functionality

Example Usage

```
const tx1 = await lucid.newTx()
   .payToAddress('addr_test...', { lovelace: 2_000_000n })
   .complete();

const tx2 = await tx1
    .chain(utxos => utxos.find(({ address }) => address ===
'addr_test...')!)
    .payToAddress('addr_test...', { lovelace: 2_000_000n })
    .payToAddress('addr_test...', { lovelace: 2_000_000n })
    .complete();
```

```diff
31 src/lucid/tx.ts
29 29    toScriptRef,
30 30    utxoToCore,
31 31  } from "../utils/mod.ts";
32    - import { applyDoubleCborEncoding } from "../utils/utils.ts";
   32 + import {
   33 +   applyDoubleCborEncoding,
   34 +   coresToUtxos,
   35 +   utxosToCores,
   36 + } from "../utils/utils.ts";
33 37  import { Lucid } from "./lucid.ts";
34 38  import { TxComplete } from "./tx_complete.ts";
35 39
36 40  export class Tx {
37 41    txBuilder: C.TransactionBuilder;
38 42    /** Stores the tx instructions, which get executed after calling .complete() */
39 43    private tasks: ((that: Tx) => unknown)[];
40    -   private lucid: Lucid;
   44 +   protected lucid: Lucid;
   45 +   /** Stores the available input utxo set for this tx (for tx chaining), if undefined falls back to wallet utxos */
   46 +   private inputUTxOs?: UTxO[];
41 47
42 48    constructor(lucid: Lucid) {
43 49      this.lucid = lucid;
@@ -91,6 +97,18 @@ export class Tx {
91 97      return this;
92 98    }
93 99
   100 +   /**
   101 +    * Defines the set of UTxOs that is considered as inputs for balancing this transactions.
   102 +    * If not set explicitely, falls back to the wallet's UTxO set.
   103 +    */
   104 +   collectTxInputsFrom(utxos: UTxO[]): Tx {
   105 +     // NOTE: merge exisitng input utxos to support tx composition
   106 +     this.tasks.push((tx) =>
   107 +       tx.inputUTxOs = [...(tx.inputUTxOs ?? []), ...utxos]
   108 +     );
   109 +     return this;
   110 +   }
   111 +
94 112    /**
```

Figure 33: src/lucid/tx.ts

31 src/lucid/tx.ts

```diff
@@ -546,13 +564,14 @@ export class Tx {
 94  112     /**
 95  113      * All assets should be of the same policy id.
 96  114      * You can chain mintAssets functions together if you need to mint assets with different policy ids.
546  564        task = this.tasks.shift();
547  565      }
548  566
549     -    const utxos = await this.lucid.wallet.getUtxosCore();
    567 +    const utxos = this.inputUTxOs !== undefined
    568 +      ? utxosToCores(this.inputUTxOs)
    569 +      : await this.lucid.wallet.getUtxosCore();
550  570
551  571      const changeAddress: C.Address = addressFromWithNetworkCheck(
552  572        options?.change?.address || (await this.lucid.wallet.address()),
553  573        this.lucid,
554  574      );
555     -
556  575      if (options?.coinSelection || options?.coinSelection === undefined) {
557  576        this.txBuilder.add_inputs_from(
558  577          utxos,
@@ -567,7 +586,6 @@ export class Tx {
567  586          ]),
568  587        );
569  588      }
570     -
571  589      this.txBuilder.balance(
572  590        changeAddress,
573  591        (() => {
@@ -602,13 +620,16 @@ export class Tx {
602  620        })(),
603  621      );
604  622
    623 +    const utxoSet = this.inputUTxOs ??
    624 +      coresToUtxos(await this.lucid.wallet.getUtxosCore());
605  625      return new TxComplete(
606  626        this.lucid,
607  627        await this.txBuilder.construct(
608  628          utxos,
609  629          changeAddress,
```

```diff
95 src/lucid/tx_complete.ts

@@ -1,27 +1,40 @@
import { C } from "../core/mod.ts";
import {
+  Credential,
  PrivateKey,
  Transaction,
  TransactionWitnesses,
  TxHash,
+  UTxO,
} from "../types/mod.ts";
+ import {
+  coresToOutRefs,
+  fromHex,
+  getAddressDetails,
+  paymentCredentialOf,
+  producedUtxosFrom,
+  toHex,
+ } from "../utils/mod.ts";
import { Lucid } from "./lucid.ts";
+ import { Tx } from "./tx.ts";
import { TxSigned } from "./tx_signed.ts";
- import { fromHex, toHex } from "../utils/mod.ts";

export class TxComplete {
  txComplete: C.Transaction;
  witnessSetBuilder: C.TransactionWitnessSetBuilder;
  private tasks: (() => Promise<void>)[];
+  /** Stores the available input utxo set for this tx (for tx chaining), if undefined falls back to wallet utxos */
+  private utxos?: UTxO[];
  private lucid: Lucid;
  fee: number;
  exUnits: { cpu: number; mem: number } | null = null;

-  constructor(lucid: Lucid, tx: C.Transaction) {
+  constructor(lucid: Lucid, tx: C.Transaction, utxos?: UTxO[]) {
    this.lucid = lucid;
    this.txComplete = tx;
    this.witnessSetBuilder = C.TransactionWitnessSetBuilder.new();
    this.tasks = [];
+    this.utxos = utxos;
```

Figure 35: src/lucid/tx_complete.ts

95 src/lucid/tx_complete.ts

```
+ /**
+  * This function provides access to the produced outputs of the current transaction
+  * that can be selectively picked to be chained with a new transaction which is returned
+  * as result.
+  *
+  * @param outputChainSelector provides the tx outputs of the transaction that can be used for chaining a new tx.
+  * If undefined is returned from this function, all outputs that are spendable from this wallet are chained.
+  * @param redeemer this arguments is expected to match the number of selected chained outputs from the first argument and can be used
+  * to chain script outputs with specific redeemers.
+  * @returns a new transaction that already has inputs set defined by the *outputChainSelector* function.
+  */
+ chain(
+   outputChainSelector: (utxos: UTxO[]) => UTxO | UTxO[] | undefined,
+   redeemer?: string | string[] | undefined,
+ ): Tx {
+   const txOutputs = producedUtxosFrom(this);
+   let chainedOutputs = outputChainSelector(txOutputs);
+   const inputUTxOs = this.getUpdatedInputUTxOs(this.utxos);
+   const chainedTx = this.lucid
+     .newTx()
+     .collectTxInputsFrom(inputUTxOs);
+
+   if (
+     !chainedOutputs ||
+     Array.isArray(chainedOutputs) && chainedOutputs.length === 0
+   ) {
+     // chain all spendable unspent transaction outputs
+     chainedOutputs = inputUTxOs;
+   }
+
+   if (Array.isArray(chainedOutputs) && Array.isArray(redeemer)) {
+     if (!redeemer || chainedOutputs.length === redeemer.length) {
+       chainedOutputs.forEach((utxo, i) =>
+         chainedTx.collectFrom([utxo], redeemer.at(i))
+       );
+     } else {
+       throw new Error(
+         `Mismatching number of chained outputs (${chainedOutputs.length}) & redeemers (${redeemer.length})`,
+       );
+     }
+   } else if (!Array.isArray(chainedOutputs) && !Array.isArray(redeemer)) {
```

95 src/lucid/tx_complete.ts

```
+    } else if (!Array.isArray(chainedOutputs) && !Array.isArray(redeemer)) {
+      chainedTx.collectFrom([chainedOutputs], redeemer);
+    } else {
+      throw new Error(
+        "Mismatching types for provided chained output(s) and redeemer(s).",
+      );
+    }
+    return chainedTx;
+  }
+
+  private getUpdatedInputUTxOs(
+    inputUTxOs?: UTxO[],
+  ): UTxO[] {
+    if (!inputUTxOs) return [];
+    const paymentCredentials = inputUTxOs.map(({ address }) =>
+      paymentCredentialOf(address)
+    );
+    const consumedOutRefs = coresToOutRefs(this.txComplete.body().inputs());
+    const isSpendableByCreds =
+      (walletPaymentCredentials: Credential[]) => ({ address }: UTxO) =>
+        walletPaymentCredentials.find(({ hash: walletPKeyHash }) => {
+          const { paymentCredential: outputPayCred } = getAddressDetails(
+            address,
+          );
+          return (outputPayCred && walletPKeyHash === outputPayCred.hash &&
+            outputPayCred.type === "Key");
+        }) !== undefined;
+    const producedUtxos = producedUtxosFrom(this);
+    const isNotConsumed = ({ txHash, outputIndex }: UTxO) =>
+      consumedOutRefs.find((outRef) =>
+        outRef.txHash === txHash && outRef.outputIndex === outputIndex
+      ) === undefined;
+    const isSpendable = isSpendableByCreds(paymentCredentials);
+    return inputUTxOs.filter(isNotConsumed).concat(
+      producedUtxos.filter(isSpendable),
+    );
+  }
  }
```

Fix memory leaks from Tx

PR Nr.	Date	Title	Submitter	Link
233	Nov 3, 2023	Fix memory leaks from Tx	joacohoyos	GitHub PR

Commits

1. Tx Leak Fix: Addressed Tx class leaks
2. Type Updates: Refined memory type handling
3. CML Optimization: Removed classes, updated protocols

Purpose

PR 233 focuses on addressing significant memory leaks within the Tx class, which were identified as a crucial issue affecting the system's performance and stability. This pull request builds on prior efforts by another contributor, yHSJ, who initiated related fixes.

Technical Changes

Comprehensive overhaul of memory handling for the Tx class, including the implementation of memory freeing routines

Refactoring of certain utility functions into a separate utilities folder to improve code organization and maintainability

Several new commits introduce enhancements and fixes across various methods within the Lucid library to ensure that memory is appropriately managed and freed

Stop Leaking Memory

PR Nr.	Date	Title	Submitter	Link
234	Nov 4, 2023	Stop Leaking Memory	yHSJ	GitHub PR

Commits

1. Memory Freeing Methods: Implemented across Data.from and Lucid class methods.
2. Code Formatting and Context: Updated freeable types and improved code formatting.
3. Extended Memory Management: Applied to Lucid.new, Lucid.switchProvider, and wallet methods.
4. Joint Enhancements: Removed CML classes, added protocol parameters, and fixed failing tests.

Purpose

PR 234 represents a comprehensive effort to address widespread memory leaks within the Lucid library and the Cardano Multiplatform Library (CML) when interfaced with JavaScript. This PR stems from extensive investigations and previous discussions about memory management issues exacerbated by WASM-JS interactions.

Technical Changes

The PR introduces enhanced memory management techniques, explicitly freeing memory for objects returned from WASM to JavaScript to prevent memory from not being garbage collected. It incorporates significant code changes across multiple utility functions and classes, ensuring that all potential memory leaks are addressed.

Related Issues

Issue Links and Background: This PR is linked to several discussions and previous PRs (e.g., 222, 233) that highlight ongoing memory management challenges