Qredit Motion Backend Wallet Integration

Specification v0.1

Michael Osullivan

Telegram: @MrMike_O

Qredit Motion uses NodeJS on it's backend processing systems. This is a guide on how to create a NodeJS Class for proper asset wallet integration into the Qredit Motion system.

Class Naming:

The file name has three parts separated by periods. First word is “wallet”, the second is the name of the coin (No spaces or special characters - all lower case). File suffix is .js

- wallet.qredit.js

Class File Structure:

In the header, put any required files you need as declared constants. If the required item needs to be from a special GitHub url instead of from Yarn package manager then make a comment with the url:

ie.

```
const got = require(‘got’);
```

Name your class the same as the coinname, with the first character upper case:

ie.

```
class Qredit {

      constructor() {

            this.hostname = null;
            …

      }

}
```

At the bottom of the file set the module exports:

ie.

```
module.exports = Qredit;
```

Class Methods:

The following methods are required in your class, We use promised responses:

setClient:

```
setClient(hostname) {
      return new Promise((resolve, reject) => {
            this.hostname = hostname;
            resolve(true);
      })
}
```

Input: Hostname or URL of client api. For example: "https://api.qredit.io/v2". This will set the hostname for later calls.

Returns: true

getNetworkInfo:

```
getNetworkInfo() {
      return new Promise((resolve, reject) => {
            (async () => {
                  // Base Response, you may add additional info
                  var inforesponse = {
                        version: “”,          (String)
                        blockheight: “”,      (Numeric String)
                        lastblock: “”         (Datetime String)
                  };
                  try {
                        … do your stuff here to populate the inforesponse
                        resolve(inforesponse);
                  } catch (e) {
                        reject(e);
                  }
            })();
      })
}
```

Input: none

Returns: inforesponse object or error

getBlock:

```
getBlock(blockheight) {
      return new Promise((resolve, reject) => {
            (async () => {
                  // Base Response, you may add additional info
                  var inforesponse = {
                        height: “”,           (Numeric String)
                        blockhash: “”,        (String)
                        blocktime: “”,        (Datetime String)
                        transactions: []      (Array of Strings)
                  };
                  try {
                        … do your stuff here to populate the inforesponse
                        resolve(inforesponse);
                  } catch (e) {
                        reject(e);
                  }
            })();
      })
}
```

Input: block height to get data on

Returns: inforesponse object or error

getTransaction:

```
getTransaction(transactionid) {
      return new Promise((resolve, reject) => {
            (async () => {
                  // Base Response, you may add additional info
                  var transinfo = {
                        totalamount: “0”,     (Numeric String)
                        blockhash: “”,        (String)
                        txid: “”,             (String)
                        fee: “”,              (Numeric String)
                        status: “”,           (confirmed, pending, error)
                        details: []           (Array of Details objects)
                  };
                  // Example Details Object
                  var details = {
                        amount: “”,           (Numeric String)
```

```
                    type: “”,             (send, receive)
                    toaddress: “”,        (String)
                    paymentid: “”         (Optional - String)
               };
               try {
                    … do your stuff here to populate the transinfo
                    resolve(transinfo);
               } catch (e) {
                    reject(e);
               }
          })();
     })
}
```

Input: Transaction ID

Returns: transinfo object

getNewAddress:

```
getNewAddress() {
     return new Promise((resolve, reject) => {
          (async () => {
               var addressinfo = {
                    address: “”,          (String)
                    privatekey: “”,       (String)
                    otherkey1: “”,        (String - Optional)
                    otherkey2: “”,        (String - Optional)
                    otherkey3: “”         (String - Optional)
               }
               try {
                    … do your stuff here to populate the addressinfo
                    resolve(addressinfo);
               } catch (e) {
                    reject(e);
               }
          })();
     })
}
```

Input: none

Returns: addressinfo object

sendTransaction:

```
sendTransaction(recipients = [], privatekeyobject) {
      return new Promise((resolve, reject) => {
            (async () => {
                  var txid = “”;
                  try {
                        … do your stuff here create the transaction
                        resolve(txid);
                  } catch (e) {
                        reject(e);
                  }
            })();
      })
}
```

Input:
- recipients: Array of Objects with address and amount.
  [{“address”:”amount”}]
- privatekeyobject: As returned from the getnewaddress call.
  {privatekey: “”, otherkey1: “”, otherkey2: “”, otherkey3:””}

Returns: String TransactionID on success, otherwise error.

Note: Some blockchains only support sending to a single address per transaction. In this case, the recipients array will contain a single item only.