SimpleCoin

This is a simplified implementation of a digital currency, modelled on Bitcoin. At present, there is only a single client (no network). This client can mine for blocks, which contain payloads. The blocks are then built into a blockchain.

Potential payloads are in a class hierarchy. A payload could be a transaction (but this subclass is currently empty) – transactions require wallets to transfer between, and wallets are not yet implemented. At present, the only payload type that is supported is a “message”, which is simply a string.

When the GUI starts, it create the genesis block of a new blockchain. The user can enter messages, which are collected in a pool. When the user starts the miner, the miner randomly selects messages from the pool (always at least 1), and mines blocks to contain these messages. The miner stops, when there are no more pending messages in the pool

As a general rule, the code is written for clarity, not for efficiency.

Block format

A block consists of a block header immediately followed by the block payload. The payload can be anything at all: transactions (if you want to simulate a coin), messages, or whatever. There is currently no maximum size, since we’re just playing around.

The block header has the following contents

Size (bytes	Content	Comments
4	Version number	Initially set to zero
32	Hash of the previous block header	This builds the blockchain
32	Hash of the payload	It’s faster to hash just this header, rather than the header and the entire payload
4	Timestamp (Unix epoch time)	When the node began looking for the hash
4	nBits threshold	The maximum acceptable value for the block hash
4	Nonce (proof of work)	May be freely set, in order to find a hash below the threshold

All hashes are generated using a single round[1] of SHA256. The block hash is generated from the contents of the header (80 bytes). If no hash is found for any value of the nonce, the timestamp is updated and the node can try again.

Integer256 and nBits

The nBits threshold uses an exponential format similar to floating point numbers, but only representing integer values. There are a variety of such formats; we use the same one that is used

1 *Bitcoin uses SHA256(SHA256()), i.e., two rounds of hashing. This is intended as a defense against attacks of the sort that reduced the security of SHA1, although no actual attacks are currently known.*

for BitCoin[2]. The first byte gives the total length of the number in bytes (think of this as a base-256 exponent). The next three bytes are the mantissa, giving the value of the number. The first bit of the mantissa must be zero, because it is (historically) a sign bit. Examples:

nBits	Value	nBits	Value	nBits	Value
0x0100????	0x00	0x04123400	0x12340000	0x04923456	-0x12345600
0x0101????	0x01	0x05001234	0x12340000	0x04123456	0x12345600
0x021234??	0x1234				
0x031234ab	0x1234ab				
0x041234ab	0x1234ab00				

Notes:

- The mantissa is not justified, so the representations of values are not unique.
- Negative numbers are not complemented; however, they are also never used in this application.
- We support this format indirectly, through the class Integer256, which represents a 256 bit integer. Such integers can be created from the nbits representation, and can also be exported to this representation.

In order to make working with hashes and the nBits format easier, SimpleCoin includes the class Integer256. An object of this class represents a 256 bit value, and the class can convert between several representations including the nBits representation.

2 *https://bitcoin.org/en/developer-reference#target-nbits*