Protocol Specification

Summary

Counterparty is a suite of financial tools in a protocol built on top of the Bitcoin blockchain and using the blockchain as a service for the reliable publication and timestamping of its messages.

The reference implementation is `counterparty-lib`, which is hosted at GitHub.

This document describes exclusively the latest version of the Counterparty protocol. For historical protocol changes, see the counterparty-lib ChangeLog.

Transactions

Counterparty messages have the following components:

- Source addresses
- Destination addresses (optional)
- A quantity of bitcoins sent from the sources to the destinations, if it exists.
- A fee, in bitcoins, paid to the Bitcoin miners who include the transaction in a block.
- Some 'data', imbedded in specially constructed transaction outputs.

Every Bitcoin transaction carrying a Counterparty transaction has the following possible outputs:

- zero or more destination outputs,
- zero or more data outputs, and optional change outputs.

All data outputs follow all destination outputs. Change outputs (outputs after the last data output) have no significance.

For identification purposes, every Counterparty transaction's 'data' field is prefixed by the string `CNTRPRTY`, encoded in UTF-8. This string is long enough that transactions with outputs containing pseudo-random data cannot be mistaken for valid Counterparty transactions. In testing (i.e. using the TESTCOIN Counterparty network on any blockchain), this string is 'XX'.

Counterparty data may be stored in three different types of outputs, or in some combinations of those formats. All of the data is obfuscated by ARC4 encryption using the transaction identifier (TXID) of the first unspent transaction output (UTXO) as the encryption key.

Multi-signature data outputs are one-of-three outputs where the first public key is that of the sender, so that the value of the output is redeemable, and the second two public keys encode the data, zero-padded and prefixed with a length byte.

The data may also be stored in `OP_RETURN` outputs or as fake pubkeyhashes.

The existence of the destination outputs, and the significance of the size of the Bitcoin fee and the Bitcoins transacted, depend on the Counterparty message type, which is determined by the four bytes in the data field that immediately follow the identification prefix. The rest of the data have a formatting specific to the message type, described in the source code.

The sources and destinations of a Counterparty transaction are Bitcoin addresses, and may be either `OP_CHECKSIG` and `OP_CHECKMULTISIG` Bitcoin ScriptPubkeys.

All messages are parsed in order, one at a time, ignoring block boundaries.

Orders, bets, order matches, bet matches and rock-paper-scissor matches are expired at the end of blocks.

Non-Counterparty transactions

counterparty-lib supports the construction of two kinds of transactions that are not themselves considered Counterparty transactions:

- BTC sends
- BTC dividends to Counterparty assets

Neither of these two transactions is constructed with a data field.

mempool transactions

Always have block index = 9999999 (`config.MEMPOOL_BLOCK_INDEX`).

DB changes never persist across sessions.

Assets

All assets except BTC and XCP have the following properties:

- Asset name
- Asset ID
- Description
- Divisiblity
- Callability
- Call date (if callable)
    - may be delayed with later issuances
- Call price (if callable) (non-negative)
    - may be increased with later issuances

Newly registered asset names will be either (unique) strings of 4 to 12 uppercase Latin characters (inclusive) not beginning with ‘A’, or integers between 26^12 + 1 and 256^8 (inclusive), prefixed with ‘A’. Alphabetic asset names will carry a one-time issuance fee (by burn) of 0.5 XCP and numeric asset names will be freely available. ‘BTC’ and ‘XCP’ are the only three-character asset names. Example asset names: BBBB, A100000000000000000.

Assets may be either divisible or indivisible, and divisible assets are divisible to eight decimal places. Assets also come with descriptions, which may be up to 52 single-byte characters long and updated at any time.

Subassets

1. Subasset names must meet following requirements :
    - Begin with the parent asset name followed by a period (.)
    - Contain at least 1 character following the parent asset name and a period (.) (e.g. PIZZA.x)
    - Contain up to 250 characters in length including the parent asset name (e.g. PIZZA.REALLY-long-VALID-Subasset-NAME)
    - Contain only characters `a-zA-Z0-9.-_@!`
    - Cannot end with a period (.)
    - Cannot contain multiple consecutive periods (..)
2. A subasset may only be issued from the same address that owns the parent asset at the time of the issuance
3. A subasset may be transferred to a new owner address after initial issuance
4. A subasset has an anti-spam issuance cost of 0.25 XCP

Transaction Statuses

Offers (i.e. orders and bets) are given a status `filled` when their `give_remaining`, `get_remaining`, `wager_remaining`, `counterwager_remaining`, `fee_provided_remaining` or `fee_required_remaining` are no longer positive quantities.

Because order matches pending BTC payment may be expired, orders involving Bitcoin cannot be filled, but remain always with a status `open`.

Message Types

- Send
- Order
- BTCPay
- Issue
- Broadcast
- Bet
- Dividend
- Burn
- Cancel

###Send

A **send** message sends a quantity of any Counterparty asset from the source address to the destination address. If the sender does not hold a sufficient quantity of that asset at the time that the send message is parsed (in the sequence of transactions), then the send is filled as much as it can be.

counterparty-lib supports sending bitcoins, for which no data output is used.

###Order

An 'order' is an offer to *give* a particular quantity of a particular asset and *get* some quantity of some other asset in return. No distinction is drawn between a 'buy order' and a 'sell order'. The assets being given are escrowed away immediately upon the order being parsed. That is, if someone wants to give 1 XCP for 2 BTC, then as soon as he publishes that order, his balance of XCP is reduced by one.

When an order is seen in the blockchain, the protocol attempts to match it, deterministically, with another open order previously seen. Two matched orders are called a 'order match'. If either of a order match's constituent orders involve Bitcoin, then the order match is assigned the status 'pending' until the necessary BTCPay transaction is published. Otherwise, the trade is completed immediately, with the protocol itself assigning the participating addresses their new balances.

All orders are *limit orders*: an asking price is specified in the ratio of how much of one would like to get and give; an order is matched to the open order with the best price below the limit, and the order match is made at *that* price. That is, if there is one open order to sell at .11 XCP/ASST, another at .12 XCP/ASST, and another at .145 XCP/BTC, then a new order to buy at .14 XCP/ASST will be matched to the first sell order first, and the XCP and BTC will be traded at a price of .11 XCP/ASST, and then if any are left, they'll be sold at .12 XCP/ASST. If two existing orders have the same price, then the one made earlier will match first.

All orders allow for partial execution; there are no all-or-none orders. If, in the previous example, the party buying the bitcoins wanted to buy more than the first sell offer had available, then the rest of the buy order would be filled by the latter existing order. After all possible order matches are made, the current (buy) order is listed as an open order itself. If there exist multiple open orders at the same price, then order that was placed earlier is matched first.

Open orders expire after they have been open for a user-specified number of blocks. When an order expires, all escrowed funds are returned to the parties that originally had them.

Order Matches waiting for Bitcoin payments expire after twenty blocks; the constituent orders are replenished.

In general, there can be no such thing as a fake order, because the assets that each party is offering are stored in escrow. However, it is impossible to escrow bitcoins, so those attempting to buy bitcoins may ask that only orders which pay a fee in bitcoins to Bitcoin miners be matched to their own. On the other hand, when creating an order to sell bitcoins, a user may pay whatever fee he likes. Partial orders pay partial fees. These fees are designated in the code as `fee_required` and `fee_provided`, and as orders involving BTC are matched (expired), these fees (required and provided) are debited (sometimes replenished), in proportion to the fraction of the order that is matched. That is, if an order to sell 1 BTC has a `fee_provided` of 0.01 BTC (a 1%), and that order matches for 0.5 BTC initially, then the `fee_provided_remaining` for that order will thenceforth be 0.005 BTC. *Provided* fees, however, are not replenished upon failure to make BTC payments, or their anti-trolling effect would be voided.

Payments of bitcoins to close order matches waiting for bitcoins are done with a **BTCpay** message, which stores in its data field only the string concatenation of the transaction hashes which compose the Order Match which it fulfils.

###Issue

Assets are issued with the **issuance** message type: the user picks a name and a quantity, and the protocol credits his address accordingly. The asset name must either be unique or be one previously issued by the same address. When re-issuing an asset, that is, issuing more of an already-issued asset, the divisibilities and the issuing address must match.

The rights to issue assets under a given name may be transferred to any other address.

Assets may be locked irreversibly against the issuance of further quantities and guaranteeing its holders against its inflation. To lock an asset, set the description to 'LOCK' (case-insensitive).

Issuances of any non-zero quantity, that is, issuances which do not merely change, e.g., the description of the asset, involve a debit (and destruction) of now 0.5 XCP.

Asset descriptions in enhanced asset information schema may be of arbitrary length.

###Broadcast

A **broadcast** message publishes textual and numerical information, along with a timestamp, as part of a series of broadcasts called a 'feed'. One feed is associated with one address: any broadcast from a given address is part of that address's feed. The timestamps of a feed must increase monotonically.

Bets are made on the numerical values in a feed, which values may be the prices of a currency, or parts of a code for describing discrete possible outcomes of a future event, for example. One might describe such a code with a text like, 'US QE on 2014-01-01: dec=1, const=2, inc=3' and announce the results with 'US QE on 2014-01-01: decrease!' and a value of 1.

The publishing of a single broadcast with a textual message equal to 'LOCK' (case-insensitive) locks the feed, and prevents it both from being the source of any further broadcasts and from being the subject of any new bets. (If a

feed is locked while there are open bets or unsettled bet matches that refer to it, then those bets and bet matches will expire harmlessly.)

The text field may be of arbitrary length.

A feed is identified by the address which publishes it.

Broadcasts with a value of -2 cancel all open bets on the feed. Broadcasts with a value of -3 cancel all pending bet matches on the feed. (This is equivalent to waiting for two weeks after the deadline.) Broadcasts with any other negative value are ignored for the purpose of bet settlement, but they still update the last broadcast time.

###Bet

A bet is a wager that the value of a particular feed will be equal (or not equal) to a certain value — the *target value* — at the *deadline*. Bets have their wagers put in escrow upon being matched, and they are settled when the feed that they rely on passes the deadline.

Equal/NotEqual Bets cannot be leveraged. However, for two Bets to be matched, their leverage levels, deadlines and target values must be identical. Otherwise, they are matched the same way that orders are, except a Bet's *odds* are the multiplicative inverse of an order's price (odds = wager/counterwager): each Bet is matched, if possible, to the open Bet with the highest odds, as much as possible.

Target values must be non-negative, and Bet Matches (contracts) are not affected by broadcasts with a value of -1.

Bets cannot have a deadline later than the timestamp of the last broadcast of the feed that they refer to.

Bets expire the same way that orders do, i.e. after a particular number of blocks. Bet Matches expire 2016 blocks after a block is seen with a block timestamp after its deadline.

Betting fees are proportional to the initial wagers, not the earnings. They are taken from, not added to, the quantities wagered.

- Because of the block time, and the non-deterministic way in which transactions are ordered in the blockchain, all contracts must not be incrementally settled, but the funds in question must be immediately put into escrow, and there must be a settlement date. Otherwise, one could see a price drop coming, and 'fight' to hide the funds that were going to be deducted.

Feed fees are deducted from the final settlement amount.

###Dividend

A dividend payment is a payment of some quantity of any Counterparty asset (including BTC) to every holder of a an asset (except BTC or XCP) in proportion to the size of their holdings. Dividend-yielding assets may be either divisible or indivisible. A dividend payment to any asset may originate from any address. The asset for dividend payments and the assets whose holders receive the payments may be the same. Bitcoin dividend payments do not employ the Counterparty protocol and so are larger and more expensive (in fees) than all other dividend payments.

- TODO: dividends on escrowed funds

There is a small fee per recipient with dividends, to prevent SPAM.

###Burn

Balances in Counterparty's native currency, 'XCP', will be initialised by 'burning' bitcoins in miners' fees during a particular period of time using the a **burn** message type. The number of XCP earned per bitcoin is calculated thus:

```
XCP_EARNED = BTC_BURNED * (1000 * (1 + .5 *
             ((END_BLOCK - CURRENT_BLOCK) / (END_BLOCK - START_BLOCK))
             ))
```

`END_BLOCK` is the block after which the burn period is over (**block #283810**) and `START_BLOCK` is the block with which the burn period begins (**block #278310**). The earlier the burn, the better the price, which may be between 1000 and

1500 XCP/BTC.

Burn messages have precisely the string 'ProofOfBurn' stored in the `OP_RETURN` output.

- new data-less burn
- burn period is over

###Cancel

Open offers may be cancelled, which cancellation is irrevocable.

A *cancel* message contains only the hash of the Bitcoin transaction that contains the order or bet to be cancelled. Only the address which made an offer may cancel it.

Counterparty Contracts

By implementing Ethereum's entire smart contracts platform Counterparty enables users to write Turing Complete smart contracts into the Bitcoin blockchain and execute those contracts in a completely decentralized and trustless manner.

Counterparty contract language is fully compatible with Ethereum's with the exception of the following minor incompatibilities:

- Two EVM opcodes (COINBASE and GASLIMIT) involved in mining were removed because XCP is unmined.
- The new ASSET_BALANCE opcode may be used to retrieve the balance of native Counterparty assets and BTC. It takes two inputs (address and asset_id) and returns one value (the balance of the address in the asset named). It has the same gas cost as BALANCE (which looks only at XCP).
- The new SEND opcode may be used for sending native Counterparty assets to Counterparty (Bitcoin) addresses. SEND has three inputs (address, quantity, asset_id) and no outputs; it has the same cost as CALL.

The basic fee structure of Counterparty Contracts is very similar to that of Ethereum. Different computational or storage operations will be associated with different fees, to prevent abuse of the system. Contract execution fees will be paid only in XCP, the native currency of Counterparty (it would not be possible for them to be paid in Bitcoin). The contract system will be fully compatible with the existing Counterparty asset system and decentralized exchange.

The economics of the fee system for Counterparty Contracts are necessarily rather different from those of Ethereum, simply because there are no Counterparty miners. All Counterparty nodes will execute all contracts, and it will be the holders of XCP that receive the fees for the execution. The simplest and most robust way to make this payment will be just to destroy the fees, and to thereby reduce the money supply, as this is equivalent to paying the fee out to all holders of XCP in proportion to the size of their holdings.

Unlike with Ethereum, the fees will not be constant values, but rather fractions of the total extant supply of XCP, so that no amount of computation will deplete the supply of XCP and drive it into negative territory: the divisibility of XCP ensures that there will always be enough XCP.

Further reading on the Ethereum contract language(s) is available here:

- Ethereum White Paper
- Ethereum Yellow Paper
- Pyethereum and Serpent Programming Guide
- Ethereum Wiki: Serpent