Offer fields:

* FeeTx
  + Contains the contents of the entire fee transaction or the promotional signed message.
  + Fee tx:
    - **offerFeeUSD** value worth of coin used to fund the fee.
    - Contains full raw tx
    - Rule - The fee tx must be in a valid block or in mempool - which is determined by asking a full node.
    - Contains a hash of the pubKey, sellAsset and buyAsset in an op\_return output
  + Signed message:
    - Instead of a real fee tx, the locked-in fields and current time signed by the LocalBazaar server with a particular private key, is also a valid “fee tx” substitute
    - Used for promotion and on-boarding for initial free offers
* feeTxCoin
  + The coin being used to pay the fee tx
  + Must be in **feeTxCoins**
  + Can also be “freeOffer” if the fee tx is just a signed message from the LocalBazaar server authorising the offer.
* tradeSizeMax
  + Measured in sell asset because price might change of buy asset over course of offer
  + Should/can be adjusted as people take small parts of your offer bit by bit
* tradeSizeMin
  + Measured in sell asset because price might change of buy asset over course of offer
* fixedPrice
  + Fixed price for the asset on offer. Optional.
  + Specified as a floating point number and will not change unless updated by the user
  + Calculated in fiat terms always e.g 10,000USD/BTC
* relativePrice
  + Left blank if fixedPrice is user. One must be emtpy
  + Specified in relation to the current price as a % premium to add or subtract e.g -0.1
  + Calculated in fiat terms always e.g 10,000USD/BTC.
  + Left blank if not dynamic
* sellAsset
  + Asset being offered
  + Locked into feeTx
  + Can be any out of a supported list of different fiats, metals and cryptos
  + Either the buyAsset or sellAsset must be a fiat currency for simplicity reasons
* buyAsset
  + Asset being bought
  + Locked into feeTx
  + Can be any out of large list of different fiats, metals and cryptos
  + Either the buyAsset or sellAsset must be a fiat currency for simplicity reasons
* coordinateLong
  + very approximate
  + Can be changed by user over the course of the trade depending on the location of the seller
* coordinateLat
  + very approximate
  + Can be changed by user over the course of the trade depending on the location of the seller
* pubKey
  + Offerer’s public key used to sign the offer
  + Locked into feeTx
  + Used to prevent maleating of offer by relaying nodes
  + And to prevent fee tx from being used by other people
* versionNumber
  + Incremented every time offer is edited so nodes know which version is newer
  + Limited to **offerEditCountMax** edits in order to prevent spam edits since they get flooded in network (it has a max value of **offerEditCountMax**)
  + Last edit reserved for deletion (effective max edits is **offerEditCountMax** - 1)
* description
  + Free text field populated by the offerer.
  + Might include trade details, Available times, Requirements, max/min amounts, links to profiles on other platforms (fwiw)
* contactMethod
  + Such as phone, signal, wire, telegram, ricochet etc.
  + Used for organising trade location, time and details between parties
* timeoutHours
  + Number of hours after the creation of the fee Tx until the offer times out
  + Counted since feeTx date
  + Specified in hours
  + Must be <= **offerAgeMaxHours**
* signature
  + Signature of hash of all fields besides signature
  + prevents malleability
  + proves ownership of fee tx
  + New private key is used for each offer
  + Priv key needed to authorise editions to offer

Variables:

* Offer Management
  + **offerFeeUSD**: 10
  + **offerSizeMaxKB**: 2kb
  + **offerEditCountMax**: 20
  + **offerAgeMaxHours**: 7 days \* 24h/day = 168h
  + **offerCheckFrequencySeconds**: 300
  + **feeTxCoins**: BTC, freeOffer
  + **txFeeSatsPerByteMin:** 100
* Peer Management
  + **peerCountThresholdMax**: 50
  + **peerCountThresholdMin**: 20
  + **peerMaxInactivityHours**: 24
  + **peerInteractionCountThresholdPerHour**: 20

MVP desktop node offer validation:

Executed by a node to validate a received offer.

There are two stages to ‘receiving’ and offer:

1. When receiving it from another node (but not yet saved to the node’s offer DB)
2. When writing it to the node’s offer database.

**The following rules are applied when receiving it from another node (but not yet saved to the node’s offer DB):**

1. Offer size <= **offerSizeMaxKB**
2. Offer hash is new, not already in offer DB.
3. All necessary fields exist.
4. One of staticPrice or relativePrice are blank.
5. Either buyAsset or sellAsset is a fiat currency. (MVP)
6. versionNumber <= **offerEditCountMax**
7. timeoutHours <= **offerAgeMaxHours**
8. tradeSizeMin <= tradeSizeMax
9. Data validation of (staticPrice OR relativePrice), tradeSizeMax, tradeSizeMin, coordinateLong and coordinateLat fields for correct data types and value ranges.
10. Fee tx is valid
    * feeTxCoin is in **feeTxCoins**
    * If feeTxCoin is “freeOffer”:
      + Check signature against localBazaar public key
    * Else:
      + Check that fee tx is either in the mempool or is confirmed in a block
      + If tx is in mempool:
        1. feeTxTxFee/feeTxSize => **txFeeSatsPerByteMin**
      + Check that coin amt is correct based off highest coin price in the two days on either side of the fee tx block time to allow for block time inconsistencies
    * Not older than timeoutHours
11. Hash of the locked-in fields match the locked-in fields hash stored in the fee tx
12. feeTx.creationTime + timeoutHours < currentTime
13. Signature is valid

**The following rules are applied when writing an offer to the node’s offer database:**

1. If don’t already have an offer with the same fee tx, save this new offer to the offer list.
2. If already have offer with the same fee tx but < versionNumber then Replace old offer with this offer.
3. If already have an offer with same fee tx but >= versionNumber, then discard this offer.

P2P offer propagation:

* Every **offerCheckFrequencySeconds**,a node will randomly select two peers from its peer list and from one, pull offers from it, and from the other, push offers to it.
* The pulling of offers consists of asking the peer to return any offers who’s hashes are not in a list of offer hashes provided.
* The push process consists of the node requesting that the peer makes a pull offer request to the node.
* Every time an interaction occurs between the node and a peer, the lastSeen time is updated for that peer in the node’s peer table
* When a new offer is found either from a peer pushing it to the node or via a query to another peer for offers, it will be saved in a temporary queue in a table before offer validation is applied for it to be added to the offer table in order to prevent conflicting offers being added asynchronously to the table.

Peer management:

* Nodes maintain a table of peers with the following attributes:
  + peerIP
  + peerRemotePort
  + lastInteractionTime
* If a peer has not been able to be contacted for **peerMaxInactivityHours**, it is removed from the node’s peer table.
* Maintaining minimum count of peers - If the amount of peers in a node’s peer table falls below the **peerCountThresholdMin**, the node will ask a current peer for X peers that they know that this node does not know. The rules are -
  + Selecting peers to ask - randomly (but it could be sequentially based on peerCountThresholdMin - but it doesn’t matter.)
  + A peer is to return a list of peers not currently known to the node. The peer selects those peers to return to the node based on the most recent **lastInteractionTime**.
* Publishing your identity to peers - A node’s peers don’t necessarily know about the node. The benefit of pushing your identity to your peers is not a personal benefit but a benefit to the network insofar as there needs to be peer lists and if no-one published their identity, then (essentially/possibly) no-one would know about anyone else. Therefore the node can ask its peers to register its existence. It goes like this -
* For interactions returnDeltaOffers and askMeForOffers, the node can include an indicator to request the peer to add the node to the peer’s peer list if its count of peers is less than **peerCountThresholdMax**.)
* A node will decide to publish its identity based on whether it is receiving too little or too many peers interacting with it within a period of time (relative to **peerInteractionCountThresholdPerHour.)**

P2P endpoints:

* returnDeltaOffers
  + Arguments:
    - portNo
    - addToPeerList
    - knownOfferHashes
  + Description:
    - Requests that the recipient returns a list of all offers that they know of that are not represented in their hashed form in knownOfferHashes.
    - If addToPeerList==True and portNo is not blank, add the requesting node to peer list unless length of peer list has reached **peerCountThresholdMax.**
  + Response:
    - List of offer objects that the requestee possesses but that the requesting node doesn’t.
* askMeForOffers
  + Arguments
    - portNo
    - addToPeerList
  + Description:
    - Requests that the receiving node makes a request to your returnDeltaOffers endpoint on port, portNo, to synchronise its offers with yours (requesting them to pull from you, effectively pushing to them)
    - If addToPeerList==True and portNo is not blank, add the requesting node to peer list unless length of peer list has reached **peerCountThresholdMax.**
  + Response:
    - None
* getPeers
  + Arguments
    - Amt of peers to return
    - peerList
  + Description:
    - Requests that receiving node responds with a list of its peers, this allows for them to share peers with each other to stay well connected to the network.
  + Response:
    - List of peers that the requestee node selected from the list of peers it is aware of

Offer culling and new offer processing loop:

* Runs in the background and both deletes offers in the table that have expired (their timeout date is prior to the current time) and also adds new offers to the table that have been queued in the offer queue table
* Every x seconds it will:
  + Query offer table for any offers with unix timestamp timeout dates prior to the current time and delete them
  + Iterate through any offers found in offer queue table and add them to the offer table after ensuring they are not duplicates and are newer versions than any current offers which use the same fee tx and then remove them from the queue

Offer creation:

* FeeTx
  + FeeTx:
    - App requests user to send **offerFeeUSD** of coins to a generated address (will provide coin amount)
      * Price of coin calculated by making a request to an exchange for the last price
    - On detection of valid incoming tx, the app will create the fee tx that spends the incoming coins, insert it into the offer and publish the offer onto the network
    - If invalid incoming tx is detected (i.e it doesn’t have the correct amount of Bitcoin), the app will broadcast a tx that returns the coins back to the sending address or ask the user to enter a new address to return the coins to and will continue to wait for a valid tx to be sent.
    - For every offer, a new intermediary address will be generated which will be derived from the private key used to sign the offer. The private key will then be saved into a data file with the corresponding feeTxID of the offer specified for each key.
* Fee tx coin
  + The user will be promoted with a selection of coins to pay the fee tx with, initially only Bitcoin will be accepted
* Max AMT
  + The user will be able to set a max amount of the asset they want to sell that they are willing to sell (e.g max 10 BTC)
* Min AMT
  + The user will be able to set a min amount of the asset they want to sell that they are willing to sell (e.g min 1 BTC)
* Price
  + The user will be able to either specify a fixed price or use the “p” variable to specify a formula to calculate a dynamic price with (e.g 2p would mean that the price will be double the current price of the asset being sold)
* Sell asset:
  + The user will be able to select what asset they want to sell in the offer
* Buy asset:
  + The user will be able to select what asset they want to buy
  + in the offer
* Approx coordinates (very approximate)
  + The user will be able to choose on a map their location which will be converted to coordinates
* Public key of sender
  + The app will generate a pub/priv key pair for each of its offers that it creates and will save them automatically to a persistence file
* Sequence number
  + Initially the sequence number will start at 0 on offer creation and then every edit that is made to the offer will increase it by 1 up to the max edit count **minus 1 (to allow for deletions by changing the timeout attribute)**
  + The user will also be made aware that there are a limited number of edits that can be made to an offer in order to reduce spam
* Description
  + The user will be able to enter any description for the offer they want and might be assisted with a hint such as “can be used to specify times of day, locations or other trade details”.
* Contact method
  + User will be able to specify a means by which people can contact them regarding their offer such as telegram, email or phone
* Timeout
  + By default this will be automatically set to 14 days (maximum life of offer) although it will be used as a way to delete offers as users can delete offers by making the timeout date < current date
* Signature
  + The signature will automatically be calculated by the app using the stored private key corresponding to the public key specified in the offer

**Immediate offer Broadcasting:**

* A user can reliably close the app once a node has had a certain number of peers pull its offer list from it that contains the new offer, either as a result of a request for them to do so or them doing so at their leisure.

Offer deletion:

* Edit offer to make timeoutHours = 0 and broadcast to network
* Offers must have one edition saved in case user wants to delete offer (**offerEditCountMax - 1)**

Paying fee:

Price data logging for fee txs:

Frontend functions:

Settings:

User journeys:

Bootstrap nodes:

Free feeTx requesting:

User stories:

* As a node I want to broadcast my existence to other nodes
* As a node I want to refresh my offers list
* As a node I want to obtain new peers up to the minimum threshold
* As a network I want there to be a perpetually available seed node
* As a node I want to connect to the network
* As a node I want to publish a new offer onto the network
* As a node I want to check the currency of my peers
* As a user I want to create an offer
* As a user I want to edit an offer
* As a user I want to delete an offer
* As a user I want to accept an offer
* As a user I want to search for offers with filters

Terms

|  |  |  |
| --- | --- | --- |
| **Term** | **Meaning** | **Notes** |
| Locked-In Fields | These are the fields in an offer what are not to be edited in future versions of the offer. i.e. they’re locked-in to the offer.  The fee tx stores a hash of these locked-in fields to ensure the system has a way to ensure these locked-in field values remain unchanged for the life of the offer. | These fields are -   * **PublicKey** - offerer’s public key used to sign the offer. * **SellAsset** - code of the asset being sold. * **BuyAsset** - code of the asset being bought. |
|  |  |  |
|  |  |  |

Future Additions:

* Messages
  + Each node will only relay max (~50) messages per offer and every message includes proof of offer tx + public key of receiver for deletion purposes
    - Nodes might limit the amount of messages per offer according to the size of the fee paid
* Account age via bch blockchain commitment
* Ability to add "contacts" or trusted users based on their public key
* Password protection
* Wallet export/import functionality
* Tor integration
* Lite client support in full nodes/development of lite client
* Pay more fees for longer timeout
* Blocking peers
* Support monero + bitcoin cash for fee tx eventually
* Deterministic priv key generation for every offer based off exportable seed.
* username reserved by putting it into block and then nodes see which one was earlier
* Commit locked fields to fee tx using merkle root
* Offer compression for bandwidth savings
* Network dynamically sharded via market and location
  + <https://gitlab.com/kadence/kadence> seems like a good node.js dht lib