Tick Liquidity

TickLiquidity structs are used to store liquidity within the Dex. Each tick has a specific price and holds liquidity for a single token.TickLiquidity come in two general types –PoolReserves for storing LP positions andLimitOrderTranches for storing maker limit orders. Both types of ticks share several common fields: PairID, TokenIn, TickIndex. PairID refers to the trading pair for which a given tick is used. TokenIn denotes which side of the TradingPair a tick holds liquidity for.

Pool Reserves

PoolReserves are the fundamental building block for Neutron DEX's AMM design. EachPoolReserves instance represents a single side of a liquidity pool. In addition to thePairID ,TokenIn andTickIndex fields, Pools Reserves also have aReserves andFee field.

type PoolReserves struct { PairID * PairID TokenIn string TickIndex int64 Reserves sdk . Int Fee uint64 } Reserves is used to store the total amount ofTokenIn within a givenPoolReserves instance andFee is the portion of the trading price that will be return to the pool.

In the context of LP liquidity,PoolReserves exist in reciprocal pairs with one side (the LowerTick) holding Token0 and the other side (the UpperTick) holding token1. Each of these pairs makes up a single constant price liquidity pool. Within each liquidity pool the following two invariants will always hold true:

1. Both PoolsReserves within a pair will have the same fee:L
2. o
3. w
4. e
5. r
6. T
7. i
8. c
9. k
10. .
11. F
12. e
13. e
14. =
15. =
16. U
17. p
18. p
19. e
20. r
21. T
22. i
23. c
24. k
25. .
26. F
27. e
28. e
29. LowerTick.Fee == UpperTick.Fee
30. L
31. o
32. w
33. er
34. T
35. i
36. c
37. k
38. .
39. F
40. ee
41. ==
42. U
43. pp
44. er
45. T
46. i

47. c
48. k
49. .
50. F
51. ee
52. The distance between the tick indexes will be equal to 2x the fee:L
53. o
54. w
55. e
56. r
57. T
58. i
59. c
60. k
61. .
62. T
63. i
64. c
65. k
66. l
67. n
68. d
69. e
70. x
71. +
72. 2
73. *
74. f
75. e
76. e
77. =
78. U
79. p
80. p
81. e
82. r
83. t
84. i
85. c
86. k
87. T
88. i
89. c
90. k
91. .
92. T
93. i
94. c
95. k
96. l
97. n
98. d
99. e
100. x
101. LowerTick.TickIndex + 2 * fee = UppertickTick.TickIndex
102. L
103. o
104. w
105. er
106. T
107. i
108. c
109. k
110. .
111. T
112. i
113. c
114. k

115. I
116. n
117. d
118. e
119. x
120. +
121. 2
122. *
123. f
124. ee
125. =
126. U
127. pp
128. er
129. t
130. i
131. c
132. k
133. T
134. i
135. c
136. k
137. .
138. T
139. i
140. c
141. k
142. I
143. n
144. d
145. e
146. x

When swaps occur the tokens will always be added and deducted within a single liqudity pool.

When LP liquidity is deposited with a given fee and price it is added to theTickLiquidity instances such that the given fee is already included in the price. For example, if Alice deposits 100 TokenA and 100TokenB at price 0 (tick 0) with a fee of 1 then her liquidity will be held in tick -1 and tick 1 respectively. If Bob were to swap 50TokenA for TokenB using Alice's liquidity his exchange rate would be~.999. His 50 TokenA would be deposited into tick -1 and he would receive 49 TokenB which would be deducted from tick 1.

It is important to note that multiplePoolReserves can exist with the same TickIndex but each one will have a unique fee.

Limit Order Tranches

LimitOrderTranches are used to store liquidity in the form of limit orders. In addition to thePairID ,TokenIn andTickIndex fields, Pools Reserves also haveTracheKey ,ReservesTokenIn ,ReservesTokenOut ,TotalTokenIn ,TotalTokenOut and an optionalExpirationTime field.

type LimitOrderTranche struct { PairID *PairID TokenIn string TickIndex int64 TrancheKey string ReservesTokenIn sdk.Int ReservesTokenOut sdk.Int TotalTokenIn sdk.Int TotalTokenOut sdk.Int ExpirationTime* time.Time } TrancheKey is a unique identifier for eachLimitOrderTranche . TrancheKeys also represent a lexicographically sortable order in which tranches with a common PairID, TokenIn and TickIndex will be traded through. Ie. A tranche with TrancheKey "A1" will be traded through before a tranche with TrancheKey "A2". ReservesTokenIn is the available token that has been added to a limit order by the "maker" and represents the amount ofTokenIn that can be traded against.ReservesTokenOut represents the filled amount of the limit order and can be withdrawn by the "maker"s.TotalTokenIn andTotalTokenOut are used to store the respective high watermarks forReservesTokenIn andReservesTokenOut and are used for the internal accounting of a limit order.

Lastly, ExpirationTime is an optional field used for Expiring limit orders (JUST_IN_TIME andGOOD_TIL_TIME). At the end of each block any LimitOrders withExpirationTime <=ctx.BlockTime() is converted to anInactiveLimitOrderTranche where it can no longer be traded against. Previous Ticks Next Liquidity Iteration