Knockout LP Calls

Knockout liquidity positions behave analogously to limit orders in traditional limit order books. At this point the sole route for users to interact with knockout liquidity is through the knockout liquidity call path. This path supports the following transaction types:

- Mint an open knockout liquidity position
- Burn an unfilled or partially filled knockout liquidity position
- Claim a fully filled knockout liquidity position
- Recover a fully filled knockout liquidity position. (Forfeits accumulated fees but requires no Merkle proof)
-

Knockout LP calls use callpath index 7:

```

Copy userCmd(7, abi.encode( code, // uint8 base, // address quote, // address poolIdx, // uint256 lowTick, // int24 highTick, // int24 isBid, // bool settleFlags, // uint8 args // bytes ))

```

The call code specifies the type of LP action according to the following codes:

- 91 - Mint knockout liquidity
- 92 - Burn open knockout liquidity
- 93 - Claim filled knockout liquidity
- 94 - Recover filled knockout liquidity
-

The remaining parameters are

- base - The address of the base token or virtual token
- quote - The address of the quote token or virtual token
- poolIdx - The arbitrary index of the pool type the user is swapping on.
- lowTick - The tick index of the lower range of the LP position
- highTick - The tick index of the upper range of the LP position
- qty - The size of the liquidity being added or removed. Fixed in terms of liquidity units, base token deposit or quote token deposit.
- settleFlags - Flag indicating how the user wants to settle the traded tokens for this swap (see Type Conventions
-)
- args - Additional ABI encoded args specific to the particular operation being called
-

Mint Calls

Mint calls create a new open knockout liquidity position. The process is similar to the classical concentrated range LP mint. Knockout LP positions impose additional restrictions compared to classical concentrated LP positions:

- The pool must explicitly enable knockout liquidity
- The width of the position (tick difference between ask and bid tick) must exactly match the pool's knockout width parameter.
- Knockout bids must be placed below the pool's current price. Knockout asks must be placed above the pool's current price.
-

The specific mint call is structured as follows:

```

Copy userCmd(7, abi.encode( 91, base, // address quote, // address poolIdx, // uint256 lowTick, // int24 highTick, // int24 isBid, // bool abi.encode( qty, // uint128 insideMid // bool ) ))

```

The additional args include:

- qty - The size of the LP positions in terms of tokens. Base tokens for bid orders (knockout liquidity below the curve price) and quote tokens for ask orders (knockout liquidity above the curve price)
- insideMid - If false, the curve's current price must be outside the order range. If true, the mint can occur with the curve price inside the range. (This should almost always be set to false.)

-

Burn Calls

Burn calls will remove the liquidity associated with an open knockout LP position. It can only be called for liquidity that hasn't previously been fully filled. It behaves similar to burning a classical concentrated LP position.

The specific burn call is structured as follows:

```

Copy userCmd(7, abi.encode( 92, base, // address quote, // address poolIdx, // uint256 lowTick, // int24 highTick, // int24 isBid, // bool abi.encode( qty, // uint128 inLiqQty, // bool insideMid // bool ) ))

```

The additional args include:

- qty - The amount of the LP position to burn in terms of tokens. Base tokens for bid orders (knockout liquidity below the curve price) and quote tokens for ask orders (knockout liquidity above the curve price)
- inLiqQty - If true qty is argument denominated in the form of sqrt(X*Y) liquidity instead of token amount.
- insideMid - If false, the curve's current price must be outside the order range. If true, the burn can occur with the curve price inside the range. True allows for burns when the order has been partially filled.
-

Recover Call

Recover calls will reclaim the converted tokens on a knockout LP position that has been fully filled and de-activated. A recover callwill not reclaim the liquidity fees accumulated by the position, however all of the underlying position will still be returned. The primary purpose to use recover instead of claim is to avoid posting the Merkle proof when accumulated fees are small relative to the cost of the calldata.

The specific recover call is structured as follows:

```

Copy userCmd(7, abi.encode( 94, base, // address quote, // address poolIdx, // uint256 lowTick, // int24 highTick, // int24 isBid, // bool abi.encode( pivotTime // uint32 ) ))

```

The additional args include:

- pivotTime - The block time that either the knockout LP tranche was minted at. (The SDK provides a function to easily query this value.)
-

Claim Call

Claim calls will reclaim the converted tokens on a knockout LP position that has been fully filled and de-activated. A claim call will reclaim the underlying filled liquidity position as well as any accumulated rewards that the position collected while in range.

The specific recover call is structured as follows:

```

Copy userCmd(7, abi.encode( 93, base, // address quote, // address poolIdx, // uint256 lowTick, // int24 highTick, // int24 isBid, // bool abi.encode( merkleRoot // uint160 merkleProof // uint96[] ) ))

```

The additional args include:

- merkleRoot - The Merkle root of the knockout tick at the time the position was minted
- merkleProof - A Merkle proof starting at the posted root that must hash to the current Merkle state of the knockout tick
-

Merkle Proof

When a given tick with active knockout LP positions is crossed, the protocol snapshots the accumulated in-range fees associated with the position. To reduce gas costs and on-chain storage requirements, each snapshot is appended as an

hashed entry to a Merkle chain rooted at the specific tick location. Successive knockout events occuring at the same tick will hash onto the previous chain's entry.

Therefore to prove the fee snapshot, the user must supply a valid Merkle proof at the time of the claim call, since the snapshot is not directly accessible on-chain. Each knockout event emits an EVM log event (CrocKnockoutCross). The event logs can be indexed off-chain to construct the Merkle proof. The SDK and Subgraph manifest provides functionality for doing this.

Previous Long Form Orders Next Pool Initialization Last updated9 months ago On this page *Mint Calls * Burn Calls * Recover Call * Claim Call