5CFE9D;}

.css-kun0x7{fill:transparent;opacity:0.5;margin:0 0.2rem;}.css-kun0x7:hover{fill:#FAF40A;}

.css-1ix0nx7{fill:transparent;opacity:0.5;}.css-1ix0nx7:hover{fill:#F14544;} On this page

Intro to Locking

The locking mechanism in V4 ensures that certain operations are executed atomically without interference, ensuring consistency and correctness in the PoolManager's state. PoolManager, usesLockers to manage a queue of lockers, ensuring that all currency deltas are settled before releasing a lock.

Pool actions can be taken by acquiring a lock on the contract and implementing thelockAcquired callback to then proceed with any of the following actions on the pools:

- swap
- modifyPosition
- donate
- take
- settle
- mint

Main Components

InPoolManager , "locking" is essentially a way to ensure certain operations are coordinated and don't interfere with each other. Here are the main components of the locking mechanism:

1.Locking and Unlocking:

- Thelock
- function is where the locking mechanism is initiated. It pushes an address tuple (address locker, address lockCaller)
- to the locker array.
- function
- lock
- (
- address
- lockTarget
- ,
- bytes
- calldata
- data
-)
- external
- payable
- override
- returns
- (
- bytes
- memory
- result
-)
- {
- Lockers
- .
- push
- (
- lockTarget
- ,
- msg
- .
- sender
-)
- ;

- // the caller does everything in this callback, including paying what they owe via calls to settle
- result
- =
- ILockCallback
- (
- lockTarget
-)
- .
- lockAcquired
- (
- msg
- .
- sender
- ,
- data
-)
- ;
- if
- (
- Lockers
- .
- length
- (
-)
- ==
- 1
-)
- {
- if
- (
- Lockers
- .
- nonzeroDeltaCount
- (
-)
- !=
- 0
-)
- revert
- CurrencyNotSettled
- (
-)
- ;
- Lockers
- .
- clear
- (
-)
- ;
- }
- else
- {
- Lockers
- .
- pop
- (
-)
- ;
- }
- }
- Copy
- During the lock, a callback functionILockCallback(lockTarget).lockAcquired(msg.sender, data)
- is called, where the locked
- contract can perform necessary operations.
- After the operations in the callback are completed, it either clears theLockers
- if it has only element,
- signifying the release of the lock, or it pops the last element fromLockers
- , signifying that the lock is

- released by that particular address.

2.Lockers.sol

TheLockers library stores an array/queue of locker addresses and their corresponding lock callers in transient storage. Each locker is represented as a tuple (address locker, address lockCaller), and each tuple occupies two slots in transient storage. The functionspush ,pop ,length , andclear are used to manage this queue.

3.Non-Zero Deltas Tracking(Flash Accounting):

- TheLockers
- library also includes a mechanism to keep track of the number of nonzero deltas
- throughnonzeroDeltaCount
- ,incrementNonzeroDeltaCount
- , anddecrementNonzeroDeltaCount
- . These functions tracks

- the number of changes or deltas that have occurred and these deltas are checked before releasing the lock.

- The_accountDelta

- function tracks the balance changes (delta) for each locker with respect to a specific currency.
- It first checks if the delta (change in balance) is zero; if it is, the function
- returns immediately as there's no change to account for. If there's a non-zero delta, the function retrieves the
- current balance for the specified locker and currency, and then calculates the new balance by adding the delta
- to the current balance.
- The function also uses theLockers
- library to increment or decrement thenonzeroDeltaCount
- that tracks the number
- of non-zero balance changes.
- function
- _accountDelta
- (
- Currency currency
- ,
- int128
- delta
-)
- internal
- {
- if
- (
- delta
- ==
- 0
-)
- return
- ;
- address
- locker
- =
- Lockers
- .
- getCurrentLocker
- (
-)
- ;
- int256
- current
- =
- currencyDelta
- [
- locker
-]
- [

- currency
-]
- ;
- int256
- next
- =
- current
- +
- delta
- ;
- unchecked
- {
- if
- (
- next
- ==
- 0
-)
- {
- Lockers
- .
- decrementNonzeroDeltaCount
- (
-)
- ;
- }
- else
- if
- (
- current
- ==
- 0
-)
- {
- Lockers
- .
- incrementNonzeroDeltaCount
- (
-)
- ;
- }
- }
- currencyDelta
- [
- locker
-]
- [
- currency
-]
- =
- next
- ;
- }
- Copy
- This mechanism is a key component of what is termed "Flash Accounting" in Uniswap V4. Flash Accounting is an
- innovative approach introduced with the new singleton-style pool management. This feature fundamentally
- alters the management of tokens during transaction processes. Traditional methods typically require explicit
- tracking of token balances at every operational phase. In contrast, Flash Accounting operates under the principle
- that by the end of each transaction or "lock period," there should be no net tokens owed either to the pool or the
- caller, streamlining the accounting process significantly.

Working

The locking mechanism in the PoolManager contract works as follows:

1. When a user wants to lock, they call thelock()
2. function with the data that they want to be passed to the callback.
3. Thelock()

4. function pushes the tuple (address locker, address lockCaller) onto the locker queue.
5. Thelock()
6. function then calls theILockCallback(lockTarget).lockAcquired(msg.sender, data)
7. callback.
8. The callback can do whatever it needs to do, such as updating the user's balances or interacting with other contracts.
9. Once the callback is finished, it returns to thelock()
10. function.
11. Thelock()
12. function checks if there are any other lockers in the queue. If there are, it pops the next locker off the queue and calls the callback for that locker.
13. If there are no more lockers in the queue, thelock()
14. function returns.

Example

Below is the example from a community"Liquidity Bootstrapping Hook" hook that uses the locking mechanism and calls themodifyPosition andswap functions.

/// @notice Callback function called by the poolManager when a lock is acquired /// Used for modifying positions and swapping tokens internally /// @param data Data passed to the lock function /// @return Balance delta function

lockAcquired (bytes

calldata data)

external override poolManagerOnly returns

(bytes

memory)

{ bytes4 selector = abi . decode (data [: 32] ,

(bytes4)) ;

if

(selector == IPoolManager . modifyPosition . selector)

{ ModifyPositionCallback memory callback = abi . decode (data [32 :] ,

(ModifyPositionCallback)) ;

BalanceDelta delta

poolManager . modifyPosition (callback . key , callback . params ,

bytes ("")) ;

if

(callback . params . liquidityDelta <

0)

{ // Removing liquidity, take tokens from the poolManager _takeDeltas (callback . key , delta , callback . takeToOwner) ;

// Take to owner if specified (exit) }

else

{ // Adding liquidity, settle tokens to the poolManager _settleDeltas (callback . key , delta) ; }

return abi . encode (delta) ; }

if

(selector == IPoolManager . swap . selector)

{ SwapCallback memory callback = abi . decode (data [32 :] ,

(SwapCallback)) ;

BalanceDelta delta

poolManager . swap (callback . key , callback . params ,

bytes ("")) ;

// Take and settle deltas _takeDeltas (callback . key , delta ,

true) ;

// Take tokens to the owner _settleDeltas (callback . key , delta) ;

return abi . encode (delta) ; }

return

bytes ("") ; } Copy Other important thing to note is that before the lock is released, thenonzeroDeltaCount is checked to ensure that all currency deltas are settled. This is done by_takeDeltas and_settleDeltas functions.

/// @notice Helper function to take tokens according to balance deltas /// @param delta Balance delta /// @param takeToOwner Whether to take the tokens to the owner function

_takeDeltas (PoolKey memory key , BalanceDelta delta ,

bool takeToOwner)

internal

{ PoolId poolId = key . toId ( ) ; int256 delta0 = delta . amount0 ( ) ; int256 delta1 = delta . amount1 ( ) ;

if

(delta0 <

0)

{ poolManager . take (key . currency0 , takeToOwner ? owner [poolId]

:

address (this) ,

uint256 (- delta0)) ; }

if

(delta1 <

0)

{ poolManager . take (key . currency1 , takeToOwner ? owner [poolId]

:

address (this) ,

uint256 (- delta1)) ; } }

/// @notice Helper function to settle tokens according to balance deltas /// @param key Pool key /// @param delta Balance delta function

_settleDeltas (PoolKey memory key , BalanceDelta delta)

internal

{ int256 delta0 = delta . amount0 ( ) ; int256 delta1 = delta . amount1 ( ) ;

if

(delta0

0)

{ key . currency0 . transfer (address (poolManager) ,

uint256 (delta0)) ; poolManager . settle (key . currency0) ; }

if

(delta1

0)

{ key . currency1 . transfer (address (poolManager) ,

uint256 (delta1)) ; poolManager . settle (key . currency1) ; } } Copy Previous Swap Tokens Next Hook Deployment * 1.Locking and Unlocking: * 2.Lockers.sol * 3.Non-Zero Deltas Tracking(Flash Accounting): * Working