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

PoolManager

To understand the major parts of the PoolManager, let's look at the the main interface:IPoolManager.sol .

https://github.com/Uniswap/v4-core/blob/main/src/interfaces/IPoolManager.sol

Lifecycle Functions

1. initialize
2. : This function initializes a new pool with specified parameters such as pool key, initial price, and
3. optional hook data, setting up the fundamental characteristics of the pool.
4. swap
5. : Allows users to exchange one type of currency for another within a specified pool, adhering to set limits
6. and conditions, and adjusting the pool's state accordingly.
7. modifyLiquidity
8. : Enables users to change the amount of liquidity they've provided to a pool, either by adding or
9. removing it, based on specified upper and lower tick limits.

Balance Functions

1. mint (related to ERC6909 claims)
2. : Used for creating new claim tokens (as per ERC6909 standards) for a user,
3. denoting specific rights or entitlements, but not representing liquidity provider (LP) receipt tokens.
4. burn (related to ERC6909 claims)
5. : Allows users to destroy their claim tokens (compliant with ERC6909),
6. effectively relinquishing the rights or entitlements those tokens represented.
7. take
8. : This function allows users to withdraw or "net out" a specified amount of a currency, which could be seen
9. as a mechanism for zero-cost flash loans under certain conditions.
10. settle
11. : Used by users to pay off any outstanding amounts they owe, potentially in a different currency, with
12. the function returning the amount paid.

Themint andburn functions are specifically related to handling ERC6909 claims, which are distinct from liquidity provider receipt tokens. These functions deal with specific claims or entitlements rather than the representation of a user's share in the liquidity pool.

Pool Initialization

Theinitialize function sets up a new liquidity pool in Uniswap. It takes necessary information such as currencies and pricing info, and hook information as inputs, checks various conditions to ensure that the pool is set up correctly, and sets initial values for the pool.

https://github.com/Uniswap/v4-core/blob/main/src/PoolManager.sol

// SPDX-License-Identifier: BUSL-1.1 pragma

solidity

^ 0.8.20 ;

// ... [other imports and contract definitions]

/// @notice Holds the state for all pools contract

PoolManager

is IPoolManager , Fees , NoDelegateCall , ERC6909Claims { // ... [other definitions and functions]

/// @inheritdoc IPoolManager function

```
initialize ( PoolKey memory key ,

uint160 sqrtPriceX96 ,

bytes

calldata hookData ) external override onlyByLocker // Modifier ensures only the current locker can call this function returns

( int24 tick ) { // Check if the fee specified in the key is too large if

( key . fee . isStaticFeeTooLarge ( ) )

revert

FeeTooLarge ( ) ;

// Validate tick spacing - it must be within defined min and max limits if

( key . tickSpacing

        MAX_TICK_SPACING )

revert

TickSpacingTooLarge ( ) ; if

( key . tickSpacing < MIN_TICK_SPACING )

revert

TickSpacingTooSmall ( ) ;

// Ensure the currency order is correct (currency0 < currency1) if

( key . currency0

        = key . currency1 )

revert

CurrenciesOutOfOrderOrEqual ( ) ;

// Validate the hooks contract address if

( ! key . hooks . isValidHookAddress ( key . fee ) )

revert Hooks . HookAddressNotValid ( address ( key . hooks ) ) ;

// Call before initialization hook with provided data key . hooks . beforeInitialize ( key , sqrtPriceX96 , hookData ) ;

// Convert the PoolKey to a PoolId PoolId id = key . toId ( ) ;

// Fetch protocol fee and dynamic swap fee if applicable ( ,

uint16 protocolFee )

=

_fetchProtocolFee ( key ) ; uint24 swapFee = key . fee . isDynamicFee ( )

?

_fetchDynamicSwapFee ( key )

: key . fee . getStaticFee ( ) ;

// Initialize the pool with the given parameters and receive the current tick tick = pools [ id ] . initialize ( sqrtPriceX96 ,
protocolFee , swapFee ) ;

// Call after initialization hook with the resulting data key . hooks . afterInitialize ( key , sqrtPriceX96 , tick , hookData ) ;

// Emit an event to signal the initialization of the pool with key details emit

Initialize ( id , key . currency0 , key . currency1 , key . fee , key . tickSpacing , key . hooks ) ; }
```

// ... [other functions] } Copy Upon success, the transaction announces a new pool was created by emitting anInitialize event.

PoolKey

ThePoolKey is a structure that uniquely identifies a liquidity pool by storing its details -- the two currencies involved (sorted numerically), the swap fee, tick spacing, and hooks (extra functionalities) of the pool.

It acts as a unique identifier, ensuring that each pool can be precisely specified and accessed within the code.

The liquidity for a PoolKey is unique to that pool alone

/// @notice Returns the key for identifying a pool struct

PoolKey

{ /// @notice The lower currency of the pool, sorted numerically Currency currency0 ; /// @notice The higher currency of the pool, sorted numerically Currency currency1 ; /// @notice The pool swap fee, capped at 1_000_000. The upper 4 bits determine if the hook sets any fees. uint24 fee ; /// @notice Ticks that involve positions must be a multiple of tick spacing int24 tickSpacing ; /// @notice The hooks of the pool IHooks hooks ; } Copy Since we create and pass thePoolKey to theinitialize function, and as part of the PoolKey we pass thehooks we want to use for the pool.

We can use thehooks to customize the pool to our liking.

Initialization Code

Here are the important parts of the initialization code from thePoolManagerInitializeTest contract.

https://github.com/Uniswap/v4-core/blob/main/test/PoolManagerInitialize.t.sol

contract

Deployers

{ function

deployFreshManager ( )

internal

{ manager =

new

PoolManager (500000) ; }

function

deployFreshManagerAndRouters ( )

internal

{ deployFreshManager ( ) ;

// Initialize various routers with the deployed manager. These routers likely handle // different aspects of the pool's functionality, such as swapping, liquidity modification, etc. swapRouter =

new

PoolSwapTest (manager) ; modifyLiquidityRouter =

new

PoolModifyLiquidityTest (manager) ; donateRouter =

new

PoolDonateTest (manager) ; takeRouter =

new

PoolTakeTest (manager) ; initializeRouter =

new

PoolInitializeTest (manager) ;

// This is the router that is used to initialize the pool

// ... [other routers] } }

contract

PoolManagerInitializeTest

is Test , Deployers , GasSnapshot { function

setUp ( )

public

{ deployFreshManagerAndRouters ( ) ;

(currency0 , currency1)

=

deployMintAndApprove2Currencies ( ) ;

uninitializedKey

PoolKey ({ currency0 : currency0 , currency1 : currency1 , fee :

3000 , hooks :

IHooks (ADDRESS_ZERO) , tickSpacing :

60 }) ; }

function

test_initialize_succeedsWithHooks (uint160 sqrtPriceX96)

public

{ // Assumptions tested in Pool.t.sol sqrtPriceX96 =

uint160 (bound (sqrtPriceX96 , TickMath . MIN_SQRT_RATIO , TickMath . MAX_SQRT_RATIO -

1)) ;

address

payable mockAddr =

payable (address (uint160 (Hooks . BEFORE_INITIALIZE_FLAG | Hooks . AFTER_INITIALIZE_FLAG))) ; address

payable hookAddr =

payable (MOCK_HOOKS) ;

vm . etch (hookAddr , vm . getDeployedCode ("EmptyTestHooks.sol:EmptyTestHooks")) ; MockContract mockContract =

new

MockContract ( ) ; vm . etch (mockAddr ,

address (mockContract) . code) ;

MockContract (mockAddr) . setImplementation (hookAddr) ;

uninitializedKey . hooks =

IHooks (mockAddr) ;

// Call initialize function with the uninitialized key and the specified sqrtPriceX96 int24 tick = initializeRouter . initialize (uninitializedKey , sqrtPriceX96 , ZERO_BYTES) ; (Pool . Slot0 memory slot0 , , ,)

= manager . pools (uninitializedKey . toId ( )) ; assertEq (slot0 . sqrtPriceX96 , sqrtPriceX96 ,

"sqrtPrice") ;

} } Copy Edit this page .css-1tclyyl{margin-top:1.5rem;} .css-1c3fvx8{display:-webkit-box;display:-webkit-flex;display:-ms-flexbox;display:flex;-webkit-flex-direction:row;-ms-flex-direction:row;flex-direction:row;-webkit-align-items:center;-webkit-box-align:center;-ms-flex-align:center;align-items:center;-webkit-box-pack:center;-ms-flex-pack:center;-webkit-justify-content:center;justify-content:center;} .css-1wsnqg4{font-size:1rem;padding-right:0.5rem;} Helpful? .css-y2jwfw{fill:transparent;opacity:0.5;}.css-y2jwfw:hover{fill:#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;} Previous V4 Architecture Overview Next Managing Positions * Lifecycle Functions * Balance Functions