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

Hook Deployment

Each hook is associated with a specific flag, represented as a constant within the contract. These constants are bit positions in an address. For instance, theBEFORE_INITIALIZE_FLAG is represented by a bit shift of1 << 159 , indicating it corresponds to the 160th bit in the address. When a hooks contract is deployed, its address's leading bits are inspected to determine which hooks are enabled.

The PoolManager, during initialization, calls the Hooks library to verify if the hooks are deployed at the correct addresses.

For example, a hooks contract deployed at the address0x9000000000000000000000000000000000000000 has leading bits '1001'. This configuration activates the hooks corresponding to these bits, such as the 'before initialize' and 'after add liquidity' hooks. This approach provides a compact and efficient method for encoding permissions directly into the contract's address.

This encoding indicates that two specific hooks ('before initialize' and 'after add liquidity') will be triggered. In the provided address:

- The leading1
- at the second-highest position aligns with theBEFORE_INITIALIZE_FLAG
- (bit 159), and
- The trailing1
- in the sequence1001
- aligns with theAFTER_ADD_LIQUIDITY_FLAG
- (bit 156).

The other two0 s in the sequence indicate that theAFTER_INITIALIZE_FLAG andBEFORE_ADD_LIQUIDITY_FLAG are not set.

Hex Hook Address Binary Address Description 0x8000000000000000000000000000000000000000 1000 0000... (bit 159) BEFORE_INITIALIZE_FLAG 0x4000000000000000000000000000000000000000 0100 0000... (bit 158) AFTER_INITIALIZE_FLAG 0x2000000000000000000000000000000000000000 0010 0000... (bit 157) BEFORE_ADD_LIQUIDITY_FLAG 0x1000000000000000000000000000000000000000 0001 0000... (bit 156) AFTER_ADD_LIQUIDITY_FLAG 0x0800000000000000000000000000000000000000 0000 1000... (bit 155) BEFORE_REMOVE_LIQUIDITY_FLAG 0x0400000000000000000000000000000000000000 0000 0100... (bit 154) AFTER_REMOVE_LIQUIDITY_FLAG 0x0200000000000000000000000000000000000000 0000 0010... (bit 153) BEFORE_SWAP_FLAG 0x0100000000000000000000000000000000000000 0000 0001... (bit 152) AFTER_SWAP_FLAG 0x0080000000000000000000000000000000000000 0000 0000 1000... (bit 151) BEFORE_DONATE_FLAG 0x0040000000000000000000000000000000000000 0000 0000 0100... (bit 150) AFTER_DONATE_FLAG 0x0020000000000000000000000000000000000000 0000 0000 0010... (bit 149) NO_OP_FLAG 0x0010000000000000000000000000000000000000 0000 0000 0001... (bit 148) ACCESS_LOCK_FLAG To generate valid hook addresses based on the code provided, we focus on the leading bits that indicate which hooks are invoked. Each flag corresponds to specific leading bits in the address, as indicated by the constants provided.

Here are some example addresses based on the flags:

1. One Hook

Example 1: Just BEFORE_SWAP_FLAG

- Address:0x2000000000000000000000000000000000000000
- Leading bits: '0010...'
- Explanation: The flag for "before swap" is set by having a '1' in the 153rd bit (from the right), represented by0x2000000000000000000000000000000000000000
- in hexadecimal.

Example 2: Just AFTER_DONATE_FLAG

- Address:0x4000000000000000000000000000000000000000
- Leading bits: '0100...'
- Explanation: The flag for "after donate" is set by having a '1' in the 150th bit, indicated by0x4000000000000000000000000000000000000000
- .

2. Two Hooks

Example 3: BEFORE_SWAP_FLAG and AFTER_SWAP_FLAG

- Address:0x3000000000000000000000000000000000000000
- Leading bits: '0011...'
- Explanation: Combining flags for "before swap" and "after swap" requires setting bits 153 and 152, resulting in0x3000000000000000000000000000000000000000
- .

Example 4: BEFORE_INITIALIZE_FLAG and AFTER_INITIALIZE_FLAG

- Address:0xC000000000000000000000000000000000000000
- Leading bits: '1100...'
- Explanation: The combination of "before initialize" and "after initialize" sets bits 159 and 158, represented by0xC000000000000000000000000000000000000000
- .

library

Hooks

{ // These flags are defined using bitwise left shifts. The1 << n operation means that the binary number 1 is shifted // to the left by n positions, effectively placing a 1 at the nth bit position (counting from the right and // starting from 0). This technique is commonly used in programming to set a specific bit in a number, which can be used // as a flag. In Ethereum addresses, which are 160 bits long, these flags correspond to the leading bits because of the // high positions of the shift (e.g., 159, 158).

uint256

internal

constant BEFORE_INITIALIZE_FLAG =

1

<<

159 ;

// (Bit 159) uint256

internal

constant AFTER_INITIALIZE_FLAG =

1

<<

158 ;

// (Bit 158) uint256

internal

constant BEFORE_ADD_LIQUIDITY_FLAG =

1

<<

```
157 ;
// (Bit 157) uint256
internal
constant AFTER_ADD_LIQUIDITY_FLAG =
1
<<
156 ;
// (Bit 156) uint256
internal
constant BEFORE_REMOVE_LIQUIDITY_FLAG =
1
<<
155 ;
// (Bit 155) uint256
internal
constant AFTER_REMOVE_LIQUIDITY_FLAG =
1
<<
154 ;
// (Bit 154) uint256
internal
constant BEFORE_SWAP_FLAG =
1
<<
153 ;
// (Bit 153) uint256
internal
constant AFTER_SWAP_FLAG =
1
<<
152 ;
// (Bit 152) uint256
internal
constant BEFORE_DONATE_FLAG =
1
<<
151 ;
```

// (Bit 151) uint256

internal

constant AFTER_DONATE_FLAG =

1

<<

150 ;

// (Bit 150) uint256

internal

constant NO_OP_FLAG =

1

<<

149 ;

// (Bit 149) uint256

internal

constant ACCESS_LOCK_FLAG =

1

<<

148 ;

// (Bit 148)

/// @notice Utility function intended to be used in hook constructors to ensure /// the deployed hooks address causes the intended hooks to be called /// @param permissions The hooks that are intended to be called /// @dev permissions param is memory as the function will be called from constructors function

validateHookPermissions (IHooks self , Permissions memory permissions)

internal

pure

{ if

(permissions . beforeInitialize != self . hasPermission (BEFORE_INITIALIZE_FLAG) || permissions . afterInitialize != self . hasPermission (AFTER_INITIALIZE_FLAG) || permissions . beforeAddLiquidity != self . hasPermission (BEFORE_ADD_LIQUIDITY_FLAG) || permissions . afterAddLiquidity != self . hasPermission (AFTER_ADD_LIQUIDITY_FLAG) || permissions . beforeRemoveLiquidity != self . hasPermission (BEFORE_REMOVE_LIQUIDITY_FLAG) || permissions . afterRemoveLiquidity != self . hasPermission (AFTER_REMOVE_LIQUIDITY_FLAG) || permissions . beforeSwap != self . hasPermission (BEFORE_SWAP_FLAG) || permissions . afterSwap != self . hasPermission (AFTER_SWAP_FLAG) || permissions . beforeDonate != self . hasPermission (BEFORE_DONATE_FLAG) || permissions . afterDonate != self . hasPermission (AFTER_DONATE_FLAG) || permissions . noOp != self . hasPermission (NO_OP_FLAG) || permissions . accessLock != self . hasPermission (ACCESS_LOCK_FLAG))

{ revert

HookAddressNotValid (address (self)) ; } }

function

hasPermission (IHooks self ,

uint256 flag)

internal

pure

returns

(bool)

{ return

uint256 (uint160 (address (self)))

& flag !=

0 ; } } Copy

CREATE2

Ethereum blockchain allows you to create contracts. There are two ways to create these contracts:

1. CREATE
2. : This is the regular way. Every time you create a contract using this, it gets a new, unique
3. address (like a house getting a unique postal address).
4. CREATE2
5. : This is a advanced way. Here, you use your address, asalt
6. which is a unique number you choose, and
7. the contract's code calledbytecode
8. to create the contract. The magic ofCREATE2
9. is that if you use the same
10. fields, you'll get the same contract address every time.

UsingCREATE2 helps ensure that the hook is deployed to the exact right address.

Here's a small code that predicts the address where a contract will be deployed usingCREATE2 before actually deploying it.

bytes32 salt =

keccak256 (abi . encodePacked (someData)) ; address predictedAddress =

address (uint (keccak256 (abi . encodePacked (byte (0xff) , deployerAddress , salt , keccak256 (bytecode))))) ; Copy

Deterministic Deployment Proxy

Many developers usehttps://github.com/Arachnid/deterministic-deployment-proxy to deploy contracts to a specific address. The main feature of this project is the use of the Ethereum CREATE2 opcode, which allows for deterministic deployment of contracts. The deployment proxy also enables the same address across different networks.

Most of the chains do have the deployment proxy at0x4e59b44847b379578588920cA78FbF26c0B4956C . Seehere for more details.

Hook Deployment Code

Thehttps://github.com/uniswapfoundation/v4-template repository contains some helper utilities for deploying hooks.

Here is the code for deploying the hooks using Deterministic Deployment Proxy which is deployed at0x4e59b44847b379578588920cA78FbF26c0B4956C :

contract

CounterScript

is Script { address

constant CREATE2_DEPLOYER =

address (0x4e59b44847b379578588920cA78FbF26c0B4956C) ; address

constant GOERLI_POOLMANAGER =

address (0x3A9D48AB9751398BbFa63ad67599Bb04e4BdF98b) ;

function

setUp ( )

public

{ }

function

run ( )

public

{ // hook contracts must have specific flags encoded in the address uint160 flags =

uint160 (Hooks . BEFORE_SWAP_FLAG | Hooks . AFTER_SWAP_FLAG | Hooks . BEFORE_ADD_LIQUIDITY_FLAG | Hooks . BEFORE_REMOVE_LIQUIDITY_FLAG) ;

// Mine a salt that will produce a hook address with the correct flags (address hookAddress ,

bytes32 salt)

= HookMiner . find (CREATE2_DEPLOYER , flags ,

type (Counter) . creationCode , abi . encode (address (GOERLI_POOLMANAGER))) ;

// Deploy the hook using CREATE2 vm . broadcast ( ) ; Counter counter =

new

Counter { salt : salt } (IPoolManager (address (GOERLI_POOLMANAGER))) ; require (address (counter)

== hookAddress ,

"CounterScript: hook address mismatch") ; } } Copyhttps://github.com/uniswapfoundation/v4-template/blob/main/script/CounterDeploy.s.sol

Note: This is a Foundry script, and it won't work for hardhat.

Read more about deploying your own hookshere . 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 Lock Mechanism - Flash Accounting Next Building your own hook * 1. One Hook * 2. Two Hooks