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

Set Up Your Local Environment

One of the most common questions we get asked is what development toolset to use to build on-chain integrations with Uniswap. There's no right answer to this question but for this guide we'll recommend a common one:Node.js ,NPM andHardhat .

At the end of this guide you'll have a development environment set up that you can use to build the rest of the examples in the Guides section of the docs, or start your own integration project!

To get you started as quickly as possible, we have provided theQuick Start section below where you can clone some boiler plate and get building. To start from scratch and learn the underlying concepts, jump to theStart from Scratch section.

Quick Start

The Uniswapboilerplate repo provides a basic Hardhat environment with required imports already pre-loaded for you. You can simply clone it and install the dependencies:

git clone https://github.com/Uniswap/uniswap-first-contract-example cd uniswap-first-contract-example npm

install Copy Then hop to theLocal Node with a Mainnet Fork to complete your set up and start developing.

Start from Scratch

In the following sections, we'll walk through the steps to create the same environment set up as the boiler plate from scratch and learn the underlying concepts.

Set Up Dependencies

Node is one of the most common Javascript runtimes. For our purposes it will provide scripting we can use to compile and test our contracts. If you haven't already, install NodeJS and its package manager NPM (instructions). Once those dependencies are set up, we can initialize our project:

npm init Copy Hardhat is an Ethereum development toolset that provides a number of powerful features including Solidity compilation, testing and deployment, all in a single convenient wrapper. We'll use NPM to add Hardhat to our project:

npm

add --save-dev hardhat Copy With Hardhat installed we can invoke it to scaffold our development environment. When you first run Hardhat you'll have the option of starting with a templated Javascript or Typescript project or an empty project. Since Hardhat relies heavily on folder structure, we recommend starting with either of the templated options. Initialize Hardhat and follow the prompts to make your selection and answer yes to the follow up prompts:

npx hardhat init Copy Once the Hardhat initialization completes, take a look around at what got set up. The folder structure should be intuitive,./contracts is where you'll write your Solidity contracts,./test is where you'll write your tests and./scripts is where you can write scripts to perform actions like deploying. Out of the box, Hardhat is configured to use this folder structure so don't change it unless you know what you're doing!

Next we'll use NPM to add the Uniswap V3 contracts which will allow us to seamlessly integrate with the protocol in our new contracts:

npm

add @uniswap/v3-periphery @uniswap/v3-core Copy The Uniswap V3 contracts were written using a past version of the solidity compiler. Since we're building integrations on V3 we have to tell Hardhat to use the correct compiler to build these files. Go to the./hardhat.config.js file and change the Solidity version to “0.7.6”:

// ... module . exports =

{ solidity :

"0.7.6" , } ; Copy That’s it! You should now have a functional development environment to start building on chain Uniswap integrations. Let’s run a quick test to confirm everything is set up properly.

Compile a Basic Contract

To confirm that our environment is configured correctly we’ll attempt to compile a basic Swap contract. Create a new file,./contracts/Swap.sol and paste the following code into it (a detailed guide to this contract can be foundhere):

// SPDX-License-Identifier: GPL-2.0-or-later pragma solidity = 0.7 .6 ; pragma abicoder v2 ;

import

'@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol' ; import

'@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol' ;

contract SimpleSwap { ISwapRouter public immutable swapRouter ; address public constant DAI

=

0x6B175474E89094C44Da98b954EedeAC495271d0F ; address public constant WETH9

=

0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 ; uint24 public constant feeTier =

3000 ;

constructor (ISwapRouter _swapRouter)

{ swapRouter = _swapRouter ; }

function

swapWETHForDAI (uint256 amountIn) external returns

(uint256 amountOut)

{

// Transfer the specified amount of WETH9 to this contract. TransferHelper . safeTransferFrom (WETH9 , msg . sender ,

address (this) , amountIn) ; // Approve the router to spend WETH9. TransferHelper . safeApprove (WETH9 ,

address (swapRouter) , amountIn) ; // Note: To use this example, you should explicitly set slippage limits, omitting for simplicity uint256 minOut =

/ *Calculate min output* /

0 ; uint160 priceLimit =

/ *Calculate price limit* /

0 ; // Create the params that will be used to execute the swap ISwapRouter . ExactInputSingleParams memory params = ISwapRouter . ExactInputSingleParams ({ tokenIn :

WETH9 , tokenOut :

DAI , fee : feeTier , recipient : msg . sender , deadline : block . timestamp , amountIn : amountIn , amountOutMinimum : minOut , sqrtPriceLimitX96 : priceLimit }) ; // The call to `exactInputSingle` executes the swap. amountOut = swapRouter . exactInputSingle (params) ; } } Copy To compile all the contracts in the./contracts folder, we’ll use the Hardhat compile command:

npx hardhat compile Copy If the environment is compiled correctly you should see the message:

Compiled { x } Solidity files successfully Copy

Local Node with a Mainnet Fork

When building and testing integrations with on chain protocols, developers often hit a problem: the liquidity on the live chain is critical to thoroughly testing their code but testing against a live network like Mainnet can be extremely expensive.

Seethe SDK getting started guide for a full example on how to use forks.

With your local node up and running, you can use the--network localhost flag in tests to point the Hardhat testing suite to that local node:

npx hardhat test --network localhost Copy

Next Steps

With your environment set up you're ready to start building. Jump over to the guides section to learn about the Uniswap functions you can integrate with. Remember to add all contracts (.sol files) to the./contracts folder and their subsequent tests to the./tests folder. You can then test them against your local forked node by running:

npx hardhat test --network localhost CopyEdit 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 Overview Next Single Swaps * Set Up Dependencies * Compile a Basic Contract