Diffchecker

Comparing sensitive data, confidential files or internal emails?

Most legal and privacy policies prohibit uploading sensitive data online. Diffchecker Desktop ensures your confidential information never leaves your computer. Work offline and compare documents securely.

Download Diffchecker Desktop

Learn more

Untitled diff

− 73 removals 111 lines

```
contract UniswapV2Factory is IUniswapV2Factory {
    address public feeTo;
    address public feeToSetter;

    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(UniswapV2Pair).creationCode));

    //  Each pair is stored in a separate address and has
    //  separate fees value that are set to default ones
    //  in function `createPair`. Can be changed by `feeToSetter`
    //  authority in setter functions

    uint public defaultMintFee;
    uint public defaultSwapFee;
    mapping(address => uint) public mintFee;
    mapping(address => uint) public swapFee;

    //  Keeps pairs, having pair token addresses as an input
    mapping(address => mapping(address => address)) public getPair;
```

+ 3 additions 44 lines

```
contract UniswapV2Factory is IUniswapV2Factory {
    address public feeTo;
    address public feeToSetter;

    mapping(address => mapping(address => address)) public getPair;
    //  Contains addresses of all the pairs
    address[] public allPairs;

    //  Solidity event to be generated everytime new pair is emitted
    //  in `createPair` function
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    //  Factory constructor, works when deploying this dApp
    //  to the network or creating an instance of it
    //  feeToSetter authority is set here that will be able to
    //  change fees using setter functions
    constructor(address _feeToSetter, uint _defaultMintFee, uint _defaultSwapFee) public {
        feeToSetter = _feeToSetter;
        defaultMintFee = _defaultMintFee;
        defaultSwapFee = _defaultSwapFee;
    }

    //  Returns amount of existing pairs
    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    //  Function to create a token pair for exchange
    //  requires tokens not to be the same, not to be zero addressed
    //  and not to exist in current set of pairs
    function createPair(address tokenA, address tokenB) external returns
```

```
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns
```

```
(address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(UniswapV2Pair).creationCode;       //  Bytecode of a `UniswapV2Pair`
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));     //  `abi.encodePacked` generates bytes from given inputs of any amount
                                                                        //  `keccak256` then hashes it and returns 32 byte hash

                                                                        //  `byte` is a dynamically arranged byte array
                                                                        //  `byte32` is 32byte statically arranged byte array
        //  Assembly operator tells that the code in it's scope is
        //  is an low-level EVM instruction

        //  We need it to precompute token pair addresses before those are being deployed as a separate smart contract
        assembly {
            pair := create2(
                0,                      // Amount of ETH transferred to the address
```

```
(address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(UniswapV2Pair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
```

```
                add(bytecode, 32), // When Solidity stores data in memory, it typically includes a 32-byte prefix that stores the length of the data. So, to get the actual contract bytecode (without the length prefix), you need to skip the first 32 bytes. add(bytecode, 32) adjusts the memory pointer to point to the actual contract bytecode, skipping the length prefix.
                mload(bytecode),    // This loads the first 32 bytes from the memory location bytecode, which typically stores the length of the bytecode. So mload(bytecode) will give you the length of the contract bytecode.
                salt                // The salt is used as part of the create2 instruction to help generate a deterministic address for the newly created contract
            )
        }

        //  Deploying UniswapV2Pair having contracts address in advance
        IUniswapV2Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        mintFee[pair] = defaultMintFee;
        swapFee[pair] = defaultSwapFee;
        emit PairCreated(token0, token1, pair, allPairs.length);
    }


    //  Below are just setter functions for the fees and
    //  fee authority, only current `feeToSetter` authority is able to
```

```
        }
        IUniswapV2Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    //  execute functions
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
```

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
```

Change 13 of 13

```

    function setMintFee(address pair, uint _mintFee) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        mintFee[pair] = _mintFee;
    }

    function setSwapFee(address pair, uint _swapFee) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        swapFee[pair] = _swapFee;
    }

    function setDefaultMintFee(uint _defaultMintFee) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        defaultMintFee = _defaultMintFee;
    }

    function setDefaultSwapFee(uint _defaultSwapFee) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        defaultSwapFee =
}                                      }
```