**Explore Open Zeppelin**

**ERC20 OpenZeppelin**

Welcome back! As mentioned in the closing of our last lesson, we could absolutely continue with manually building out a smart contract comprised of all the required functions to be compatible with the ERC20 standard, but wouldn't it be more convenient to use pre-deployed, audited, and ready-to-go contracts?

In this section, I'll guide you on using the OpenZeppelin Library to achieve this.

❗ **NOTE** OpenZeppelin is renowned for its Smart Contract framework, offering a vast repository of audited contracts readily integrable into your codebase.

Access [OpenZeppelin's documentation](https://docs.openzeppelin.com/contracts/5.x/) via their official website. By navigating to [Products -> Contracts Library](https://www.openzeppelin.com/contracts), you can discover a vast array of ready-to-use contracts.

Additionally, OpenZeppelin offers a contract wizard, streamlining the contract creation process — perfect for tokens, governances, or custom contracts. <https://wizard.openzeppelin.com/#erc20>

Let's leverage OpenZeppelin to create a new ERC20 Token. Create a new file within src named OurToken.sol. Once that's done, let's install the OpenZeppelin library into our contract.

forge install OpenZeppelin/openzeppelin-contracts --no-commit

Once installed you'll see the ERC20 contract from OpenZeppelin within `lib/openzeppelin-contracts/token/ERC20/ERC20.sol`. Let's add a remapping in our foundry.toml to make importing a little easier on us.Within foundry.toml add the line:

remappings = ["@openzeppelin=lib/openzeppelin-contracts"]

We can now import and inherit this contract into OurToken.sol!

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract OurToken is ERC20 {

//constructor goes here

}

By importing the OpenZeppelin implementation of ERC20 this way, we inherit all the functionality of the ERC20 standard with much less work and a level of confidence that the code has been testing and verified.

❗ **PROTIP** If you're looking for an alternative library full of trusted contracts, I recommend looking at the [**Solmate Repo**](https://github.com/transmissions11/solmate) by Transmissions11.

Now, we should recall that when inheriting from a contract with a constructor, our contract must fulfill the requirements of that constructor. We'll need to define details like a name and symbol for OurToken.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract OurToken is ERC20 {

constructor(uint256 initialSupply) ERC20("OurToken", "OT") {

\_mint(msg.sender, initialSupply);

}

}

For the purposes of simple examples like this, I like to mint the initialSupply to the deployer/msg.sender, which I've demonstrated above.

As always we can perform a sanity check to assure things are working as expected by running forge build.

Nailed it.

See you in the next lesson where we'll look into how to deploy this bad Larry.