Solidity libraries

With Filecoin Virtual Machine (FVM), Solidity developers can use existing libraries listed on this page in their FVM smart contracts.

OpenZeppelin

OpenZeppelin provides a library of battle-tested smart contract templates, including widely used implementations of ERC token standards. For a guided example that implements an ERC20 token on the Filecoin network, seeExample using an ERC20 contract .

Benefits

OpenZeppelin offers the following to smart contract developers:

- Implementations of standards like ERC20, ERC721, and ERC1155.
- Flexible access control schemes likeOwnable
- ,AccessControl
- , andonlyRole
- .
- Useful and secure utilities for signature verification,SafeMath
- , etc..
-

Token standards, such asERC20 , are the most widely used smart contract libraries from OpenZeppelin. These contracts, listed below, implement bothfungible andnon-fungible tokens:

- ERC20
- is the simplest and most widespread token standard for fungible assets.
- ERC721
- is the standard solution for non-fungible tokens and is often used for collectibles and games.
- ERC777
- provides a richer standard for fungible tokens, supporting new use cases and backwards compatibility with ERC20.
- ERC1155
- is a new standard formulti-tokens
- , where a single contract represents multiple fungible and non-fungible tokens, and operations are batched for increased gas efficiency.
-

Using OpenZeppelin with FVM

Thegeneral procedure for using OpenZeppelin with FVM is as follows:

1. Install OpenZeppelin. For example, usingnpm
2. :
3.

```

Copy npm install @openzeppelin/contracts

```

1. Import the specific library you want to use.
2. In your smart contract, inherit the library.
3.

Thanks to the FVM, your contract can be integrated and deployed on the Filecoin network with OpenZeppelin inheritance. For a guided example that implements an ERC20 token on the Filecoin network, seeExample using an ERC20 contract .

Example using an ERC-20 contract

In the following tutorial, you'll write and deploy a smart contract that implements theERC-20 on the Calibration testnet using Remix and MetaMask:

Prerequisites

Let's take an ERC20 contract as an example to write and deploy it on the Calibration testnet using Remix & MetaMask:

- Remix.

- MetaMask.
- MetaMask connected to the Calibration testnet
- .
- Test tokens (tFIL)from the faucet
- .
-

Procedure

In this procedure, you will create, deploy, mint and send anERC20 token on Calibration using Remix and MetaMask.

1. Navigate toremix.ethereum.org
2. .
3. Next toWorkspaces
4. , click the+
5. icon to create a new workspace.
6. In theChoose a template
7. dropdown, selectERC20
8. along with theMintable
9. checkbox
10. ClickOK
11. .
12. In thecontract
13. directory, openMyToken.sol
14. .
15. Set the token
16. and
17. :
18.

```

Copy // contracts/GLDToken.sol // SPDX-License-Identifier: MIT pragmasolidity^0.8.0;

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

contractMyTokenisERC20{ constructor(uint256initialSupply)ERC20(, ) { _mint(msg.sender,initialSupply); } }

```

1. Next, compile and deploy the contract on Filecoin.
2. At the top of the workspace, click the green play symbol to compile the contract.
3. Once the contract compiles, open theDeploy
4. tab on the left.
5. Under theEnvironment
6. dropdown, selectInjected Provider - MetaMask
7. .
8. In the MetaMask popup window, selectConfirmed connection
9. .
10. ClickDeploy
11. , and confirm the transaction on MetaMask. Your token contract will be deployed to the Calibration testnet once the network confirms the transaction.
12. In Remix, open theDeployed Contracts
13. dropdown.
14. In themint
15. method, set:
16.
    - to
17.
    - to your wallet address.
18.
    - amount
19.
    - to10000000000000000000
20.
    - (1FIL
21.
    -).
22. *
23. ClickTransact

24. .
25. In MetaMask, confirm the transaction.
26.

Once the network processes the transaction, the token is minted and sent to your network address. Congratulations, you've completed the tutorial!

Additional resources

Learn more about OpenZeppelin with the following resources:

- OpenZeppelin Contracts website
- Documentation
- GitHub
-

DappSys

The DappSys library provides safe, simple, and flexible Ethereum contract building blocks for common Ethereum and Solidity use cases.

- Documentation
- GitHub
-

0x protocol

The 0x protocol library provides a set of secure smart contracts that facilitate peer-to-peer exchange of Ethereum-based assets.

- Documentation
- GitHub
-

Previous Foundry Next Call built-in actors

Last updated14 days ago