Best practices

This page describes best practices for testing, developing and deploying smart contracts on the Filecoin network.

Transactions

Best practices for transactions are described below.

Consistently generating transaction receipts

Since receipts in Filecoin are generated in the next tipset, depending on when a transaction is submitted to the mempool, the receipt may take between 30 and 90 seconds to return. To consistently return transaction receipts when deploying a transaction or awaiting confirmation, change the default transaction receipt timeout (60000 ms or 1 minute for many toolchains) to 90 seconds or more. An example that setstimeout to180000 (3 minutes) for an Open Zeppelin upgradeable proxy is as follows:

```

Copy constdeployment=awaitupgrades.deployProxy(contract,preparedArguments,{ timeout:180000 });

```

Unstuck a message from the mempool

When users send messages to the Filecoin network, those messages will first land in the mempool. Once a node receives your message, it will verify the gas fee and signature and then process the transaction. Depending on network traffic and other factors, this process may take some time.

If the Filecoin network still needs to confirm a message, it's because it has yet to be processed and is sitting in the mempool. Several causes exist, such as network congestion, insufficient gas fees, or an invalid message signature.

We recommend users resubmit the message with a higher gas fee or priority fee so those messages will not block the mempool and potentially impact the block-producing time. Gas fees on the network can fluctuate depending on network demand, so it's always a good idea to monitor gas prices and adjust your fees accordingly to ensure your transaction is processed promptly.

Metamask

If you are building your project using MetaMask, it would be easier to educate the users to speed up a transaction by increasing the gas fee directly in MetaMask. Refer to theofficial MetaMask documentation for more details.

Lotus

Developers using Lotus canreplace an existing message with an updated gas fee.

SDKs

Developers processing messages using SDKs, such as ethers.js or web3.js, must replace the message with higher gas fees by following these steps:

1. Get the original message using its hash.
2. Create a new message with the samenonce
3. ,to
4. , andvalue
5. fields as the original message.
6. Set a highergasLimit
7. andgasPrice
8. for this message.
9. Sign and send the new message.
10.

Futureproofing

Developers should take the time to thoroughly read through the following summary of possible contract future-proofing updates, as failure to properly future proof smart contracts may result in incompatibility with future Filecoin releases.

- All contracts
- mustaccept bothDAG_CBOR (0x71) andCBOR (0x51) in inputs and treat them identically, and useCBOR (0x51) in outputs

- .
- If a contract uses the FRC42 hash ofGranularityExported
- , it must be updated and redeployed.
- If a contract sends funds to actors that are non-native, Ethereum, or EVM smart contract accounts, it must use thecall_actor precompile
- .
- If a contract is interacting with built-in actors, it must upgrade to the latest version of Filecoin Solidity library, currentlyv0.8
- .
-

All contracts

All contracts must do the following:

Accept both DAG_CBOR (0x71) and CBOR (0x51) in inputs and treat them identically

Smart contracts should accept bothDAG_CBOR (0x71) andCBOR (0x51) in inputs and treat them identically. Specifically:

- TreatDAG_CBOR
- andCBOR
- as equivalent when returned from thecall_actor
- precompile.
- TreatDAG_CBOR
- andCBOR
- as equivalent when received as a parameter tohandle_filecoin_method
- .
-

Use CBOR (0x51) in outputs

Smart contracts should useCBOR (0x51) in outputs. Specifically:

- Always passCBOR
- to thecall_actor
- precompile.DAG_CBOR
- is currently forbidden.
- Always returnCBOR
- fromhandle_filecoin_method
- .DAG_CBOR
- is currently forbidden.
-

Contracts usingGranularityExported hash

TheGranularityExported method in the Datacap actor was renamed toGranularity , so any contracts which use the FRC42 hash ofGranularityExported (953701584) must update the hash to3936767397 and redeploy.

Contracts sending funds to specific actors

Any contracts sending funds to actors that are not native accounts (f1 orf3 addresses), Ethereum accounts, or EVM smart contracts must now use thecall_actor precompile.Solidity's transfer function will no longer work as that will attempt to invoke the target actor as an EVM contract .

Contracts interacting with built-in actors

All contracts interacting with built-in actors must upgrade to thelatest version of Filecoin Solidity library, currentlyv0.8 . The IPLD codec used in thehandle_filecoin_method solidity entrypoint and thecall_actor should now beCBOR (0x51) , notDAG_CBOR (0x71) , as previously used. The underlying encoding (i.e. the payload bytes) are the same, but the codec numbers are now different.DAG_CBOR support will be re-enabled in the future but the usage of the codec implies additional runtime guarantees that have not yet been implemented.

Contract Verification

When deploying contracts to mainnet, it is important to verify your contracts to improve transparency, security and trustlessness of the network. The process of verifying your contract involves recompiling your contract's source code to ensure that the produced bytecode matches the bytecode that is already live on the network since it was deployed.

It is highly recommended for all FVM smart contracts to complete the verification process, soon after deployment.

Developers can easily do so through the following block explorers:

- Filfox contract verifier
- Beryx contract verifier
-

You can find this tutorial in the FEVM ERC-20 Quickstart .

Previous Verify a contract Next Advanced

Last updated7 months ago