---

title: "Downsizing contracts to fight the contract size limit" description: What can you do to prevent your smart contracts from getting too large? author: Markus Waas lang: en tags: ["solidity", "smart contracts", "storage", "truffle"] skill: intermediate published: 2020-06-26 source: soliditydeveloper.com sourceUrl: https://soliditydeveloper.com/max-contract-size

---

# Why is there a limit? {#why-is-there-a-limit}

On November 22, 2016 the Spurious Dragon hard-fork introduced EIP-170 which added a smart contract size limit of 24.576 kb. For you as a Solidity developer this means when you add more and more functionality to your contract, at some point you will reach the limit and when deploying will see the error:

```
Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off revert strings, or using libraries.
```

This limit was introduced to prevent denial-of-service (DOS) attacks. Any call to a contract is relatively cheap gas-wise. However, the impact of a contract call for Ethereum nodes increases disproportionately depending on the called contract code's size (reading the code from disk, pre-processing the code, adding data to the Merkle proof). Whenever you have such a situation where the attacker requires few resources to cause a lot of work for others, you get the potential for DOS attacks.

Originally this was less of a problem because one natural contract size limit is the block gas limit. Obviously, a contract must be deployed within a transaction that holds all of the contract's bytecode. If you include only that one transaction into a block, you can use up all that gas, but it's not infinite. Since the London Upgrade, the block gas limit has been able to vary between 15M and 30M units depending on network demand.

# Taking on the fight {#taking-on-the-fight}

Unfortunately, there is no easy way of getting the bytecode size of your contracts. A great tool to help you that is the truffle-contract-size plugin if you're using Truffle.

1. `npm install truffle-contract-size`
2. Add the plugin to the *truffle-config.js*: `plugins: ["truffle-contract-size"]`
3. Run `truffle run contract-size`

This will help you figure out how your changes are affecting the total contract sizes.

In the following we will look at some methods ordered by their potential impact. Think about it in the terms of weight-loss. The best strategy for someone to hit their target weight (in our case 24kb) is to focus on the big impact methods first. In most cases just fixing your diet will get you there, but sometimes you need a little bit more. Then you might add some exercise (medium impact) or even supplements (small impact).

# Big impact {#big-impact}

## Separate your contracts {#separate-your-contracts}

This should always be your first approach. How can you separate the contract into multiple smaller ones? It generally forces you to come up with a good architecture for your contracts. Smaller contracts are always preferred from a code readability perspective. For splitting contracts, ask yourself:

- Which functions belong together? Each set of functions might be best in its own contract.
- Which functions don't require reading contract state or just a specific subset of the state?
- Can you split storage and functionality?

## Libraries {#libraries}

One simple way to move functionality code away from the storage is using a library. Don't declare the library functions as

internal as those will be added to the contract directly during compilation. But if you use public functions, then those will be in fact in a separate library contract. Consider using for to make the use of libraries more convenient.

### Proxies {#proxies}

A more advanced strategy would be a proxy system. Libraries use `DELEGATECALL` in the back which simply executes another contract's function with the state of the calling contract. Check out this blog post to learn more about proxy systems. They give you more functionality, e.g., they enable upgradability, but they also add a lot of complexity. I wouldn't add those only to reduce contract sizes unless it's your only option for whatever reason.

## Medium impact {#medium-impact}

### Remove functions {#remove-functions}

This one should be obvious. Functions increase a contract size quite a bit.

- **External**: Often times we add a lot of view functions for convenience reasons. That's perfectly fine until you hit the size limit. Then you might want to really think about removing all but absolutely essential ones.
- **Internal**: You can also remove internal/private functions and simply inline the code as long the function is called only once.

### Avoid additional variables {#avoid-additional-variables}

A simple change like this:

```
solidity function get(uint id) returns (address,address) { MyStruct memory myStruct = myStructs[id]; return (myStruct.addr1, myStruct.addr2); }

solidity function get(uint id) returns (address,address) { return (myStructs[id].addr1, myStructs[id].addr2); }
```

makes a difference of **0.28kb**. Chances are you can find many similar situations in your contracts and those can really add up to significant amounts.

### Shorten error message {#shorten-error-message}

Long revert messages and in particular many different revert messages can bloat up the contract. Instead use short error codes and decode them in your contract. A long message could be become much shorter:

```solidity require(msg.sender == owner, "Only the owner of this contract can call this function");

```

```
solidity require(msg.sender == owner, "OW1");
```

### Use custom errors instead of error messages

Custom errors have been introduced in Solidity 0.8.4. They are a great way to reduce the size of your contracts, because they are ABI-encoded as selectors (just like functions are).

```solidity error Unauthorized();

if (msg.sender != owner) { revert Unauthorized(); } ```

### Consider a low run value in the optimizer {#consider-a-low-run-value-in-the-optimizer}

You can also change the optimizer settings. The default value of 200 means that it's trying to optimize the bytecode as if a function is called 200 times. If you change it to 1, you basically tell the optimizer to optimize for the case of running each function only once. An optimized function for running only one time means it is optimized for the deployment itself. Be aware that **this increases the gas costs for running the functions**, so you may not want to do it.
```

# Small impact {#small-impact}

## Avoid passing structs to functions {#avoid-passing-structs-to-functions}

If you are using the ABIEncoderV2, it can help to not pass structs to a function. Instead of passing the parameter as a struct...

```solidity function get(uint id) returns (address,address) { return _get(myStruct); }

function _get(MyStruct memory myStruct) private view returns(address,address) { return (myStruct.addr1, myStruct.addr2); } ```

```solidity function get(uint id) returns(address,address) { return _get(myStructs[id].addr1, myStructs[id].addr2); }

function _get(address addr1, address addr2) private view returns(address,address) { return (addr1, addr2); } ```

... pass the required parameters directly. In this example we saved another **0.1kb**.

## Declare correct visibility for functions and variables {#declare-correct-visibility-for-functions-and-variables}

- Functions or variables that are only called from the outside? Declare them as `external` instead of `public`.
- Functions or variables only called from within the contract? Declare them as `private` or `internal` instead of `public`.

## Remove modifiers {#remove-modifiers}

Modifiers, especially when used intensely, could have a significant impact on the contract size. Consider removing them and instead use functions.

```solidity modifier checkStuff() {}

function doSomething() checkStuff {} ```

```solidity function checkStuff() private {}

function doSomething() { checkStuff(); } ```

Those tips should help you to significantly reduce the contract size. Once again, I cannot stress enough, always focus on splitting contracts if possible for the biggest impact.