---

title: Smart contract libraries description: lang: en

You don't need to write every smart contract in your project from scratch. There are many open source smart contract libraries available that provide reusable building blocks for your project that can save you from having to reinvent the wheel.

Prerequisites {#prerequisites}

Before jumping into smart contract libraries, it's a good idea to have a nice understanding of the structure of a smart contract. Head over to smart contract anatomy if you haven't done so yet.

What's in a library {#whats-in-a-library}

You can usually find two kinds of building blocks in smart contract libraries: reusable behaviors you can add to your contracts, and implementations of various standards.

Behaviors {#behaviors}

When writing smart contracts, there is a good chance you'll find yourself writing similar patterns over and over, like assigning an *admin* address to carry out protected operations in a contract, or adding an emergency*pause* button in the event of an unexpected issue.

Smart contract libraries usually provide reusable implementations of these behaviors aslibraries or via inheritance in Solidity.

As an example, following is a simplified version of the`Ownable` contract from the OpenZeppelin Contracts library, which designates an address as the owner of a contract, and provides a modifier for restricting access to a method only to that owner.

```solidity contract Ownable { address public owner;

```
constructor() internal {
    owner = msg.sender;
}

modifier onlyOwner() {
    require(owner == msg.sender, "Ownable: caller is not the owner");
    _;
}
```

} ```

To use a building block like this in your contract, you would need to first import it, and then extend from it in your own contracts. This will allow you to use the modifier provided by the base `Ownable` contract to secure your own functions.

```solidity import ".../Ownable.sol"; // Path to the imported library

contract MyContract is Ownable { // The following function can only be called by the owner function secured() onlyOwner public { msg.sender.transfer(1 ether); } } ```

Another popular example is SafeMath or DsMath. These are libraries (as opposed to base contracts) that provide arithmetic functions with overflow checks, which are not provided by the language. It's a good practice to use either of these libraries instead of native arithmetic operations to guard your contract against overflows, which can have disastrous consequences!

### Standards {#standards}

To facilitate composability and interoperability, the Ethereum community has defined several standards in the form of**ERCs**. You can read more about them in the standards section.

When including an ERC as part of your contracts, it's a good idea to look for standard implementations rather than trying to

roll out your own. Many smart contract libraries include implementations for the most popular ERCs. For example, the ubiquitous ERC20 fungible token standard can be found in HQ20, DappSys and OpenZeppelin. Additionally, some ERCs also provide canonical implementations as part of the ERC itself.

It's worth mentioning that some ERCs are not standalone, but are additions to other ERCs. For example ERC2612 adds an extension to ERC20 for improving its usability.

## How to add a library {#how-to}

Always refer to the documentation of the library you are including for specific instructions on how to include it in your project. Several Solidity contract libraries are packaged using `npm`, so you can just `npm install` them. Most tools for compiling contracts will look into your `node_modules` for smart contract libraries, so you can do the following:

```solidity // This will load the @openzeppelin/contracts library from your node_modules import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract MyNFT is ERC721 { constructor() ERC721("MyNFT", "MNFT") public { } } ```

Regardless of the method you use, when including a library, always keep an eye on the language version. For instance, you cannot use a library for Solidity 0.6 if you are writing your contracts in Solidity 0.5.

## When to use {#when-to-use}

Using a smart contract library for your project has several benefits. First and foremost, it saves you time by providing you with ready-to-use building blocks you can include in your system, rather than having to code them yourself.

Security is also a major plus. Open source smart contract libraries are also often heavily scrutinized. Given many projects depend on them, there is a strong incentive by the community to keep them under constant review. It's much more common to find errors in application code than in reusable contract libraries. Some libraries also undergo external audits for additional security.

However, using smart contract libraries carry the risk of including code you are not familiar with into your project. It's tempting to import a contract and include it directly into your project, but without a good understanding of what that contract does, you may be inadvertently introducing an issue in your system due to an unexpected behavior. Always make sure to read the documentation of the code you are importing, and then review the code itself before making it a part of your project!

Last, when deciding on whether to include a library, consider its overall usage. A widely-adopted one has the benefits of having a larger community and more eyes looking into it for issues. Security should be your primary focus when building with smart contracts!

## Related tools {#related-tools}

**OpenZeppelin Contracts -** ***Most popular library for secure smart contract development.***

- Documentation
- GitHub
- Community Forum

**DappSys -** ***Safe, simple, flexible building-blocks for smart-contracts.***

- Documentation
- GitHub

**HQ20 -** ***A Solidity project with contracts, libraries and examples to help you build fully-featured distributed applications for the real world.***

- GitHub
```

thirdweb Solidity SDK - ***Provides the tools needed to build custom smart contracts efficiently***

- Documentation
- GitHub

Related tutorials {#related-tutorials}

- Security considerations for Ethereum developers *– A tutorial on security considerations when building smart contracts, including library usage.*
- Understand the ERC-20 token smart contract *-Tutorial on the ERC20 standard, provided by multiple libraries.*

Further reading {#further-reading}

Know of a community resource that helped you? Edit this page and add it!