Lesson 19

Documentation

Natspec

See Docs

How to comment code in Solidity?
Comments in Solidity can be written in two different ways.

```
4     // I am a standard single-line comment
5     /// I am a Natspec single-line comment
6
7     /*
8      I am a standard
9      multi-line
10     comment
11    */
12
13    /**
14     I am a Natspec
15     multi-line
16     comment
17    */
```

What are Natspec Comments?
Special form of comments in Solidity contracts
⇒ Machine Readable
Used to documents variables, functions, contracts, etc...
Based on the Ethereum Natural Language Specification Format (NatSpec)

Single line Natspec comment: start with ///
Multi line Natspec comment: start with `/**`, end with `*/`

The Solidity compiler only interprets tags if they are external or public. You are welcome to use similar comments for your internal and private functions, but those will not be parsed.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 < 0.9.0;

/// @title A simulator for trees
/// @author Larry A. Gardner
/// @notice You can use this contract for only the most basic
simulation
/// @dev All function calls are currently implemented without
side effects
```

```
/// @custom:experimental This is an experimental contract.
contract Tree {
    /// @notice Calculate tree age in years, rounded up, for
live trees
    /// @dev The Alexandr N. Tetearing algorithm could increase
precision
    /// @param rings The number of rings from
dendrochronological sample
    /// @return Age in years, rounded up for partial years
    function age(uint256 rings) external virtual pure returns
(uint256) {
        return rings + 1;
    }

    /// @notice Returns the amount of leaves the tree has.
    /// @dev Returns only a fixed number.
    function leaves() external virtual pure returns(uint256) {
        return 2;
    }
}

...

    /// Return the amount of leaves that this specific kind of
tree has
    /// @inheritdoc Tree
    function leaves() external override(Tree, Plant) pure
returns(uint256) {
        return 3;
    }
}
```

What do Natspec comments do?

Document smart-contracts for developers
Generate documentation for the smart contracts automatically with third-party tools.
Annotate conditions for formal verification.

Using the @dev tag

Notify end-users when interacting with the contract
= more expressive

Show relevant details to end users at the time they will interact with the contract (= sign a transaction.

Using the @notice tag (only in public and external functions).

Tag		Context
`@title`	A title that should describe the contract/interface	contract, library, interface
`@author`	The name of the author	contract, library, interface
`@notice`	Explain to an end user what this does	contract, library, interface, function, public state variable, event
`@dev`	Explain to a developer any extra details	contract, library, interface, function, state variable, event
`@param`	Documents a parameter just like in Doxygen (must be followed by parameter name)	function, event
`@return`	Documents the return variables of a contract's function	function, public state variable
`@inheritdoc`	Copies all missing tags from the base function (must be followed by the contract name)	function, public state variable
`@custom:...`	Custom tag, semantics is application-defined	everywhere

What should we document in Natspec?

- Contracts
  Including interfaces and libraries

- Functions,
  Including constructors and public state variables (with automatic getter).

- Events

Tags

`@title`

A title that should describe the contract/interface

context : contract, library, interface

`@author`

The name of the author

context : contract, library, interface

`@notice`

Explain to an end user what this does

context : contract, library, interface, function, public state variable, event

`@dev`

Explain to a developer any extra details

context : contract, library, interface, function, state variable, event

`@param`

Documents a parameter just like in Doxygen (must be followed by parameter name)

context : function, event

`@return`

Documents the return variables of a contract's function

context : function, public state variable

`@inheritdoc`

Copies all missing tags from the base function (must be followed by the contract name)

context : function, public state variable

`@custom:...`

Custom tag, semantics is application-defined

context : anywhere

Documenting Contracts with Natspec

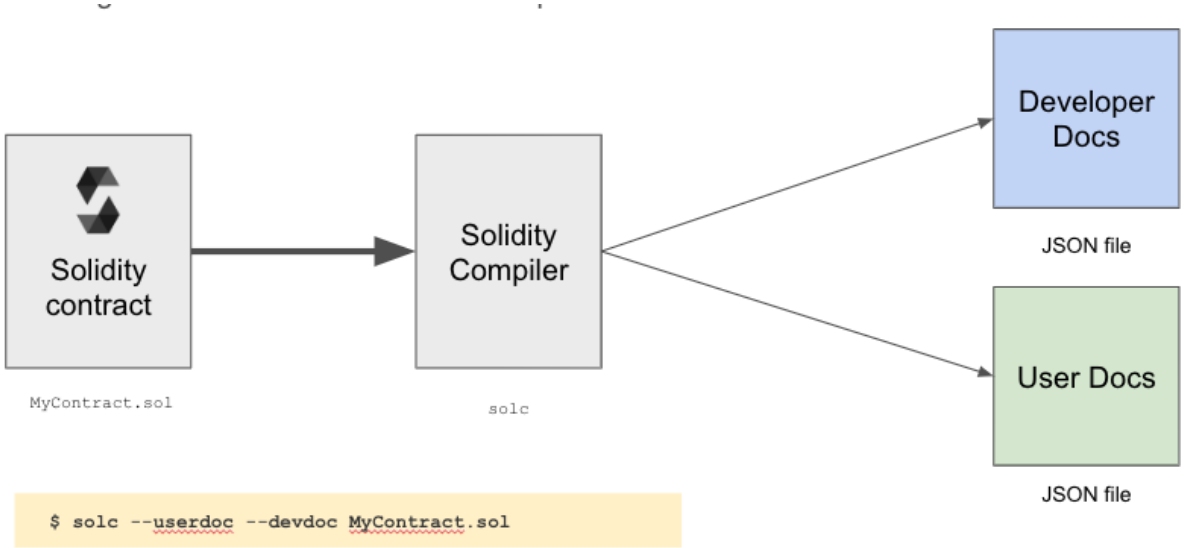
Example: Argent
Example: Buffer Library from Oraclize

@param must be followed by the variable name passed as argument.
@return good practice is to put return type or the name of the variable returned.
@notice only relevant in public + external functions (only for userdocs).

Example: Uniswap
Example: Uniswap

Example where:
A public state variable (= automatic getter function)
Inherit the docs of the parent / base contract (= here the interface)

Documentation Generator



The Solidity compiler generates a JSON file
= artifacts with contract metadata, that contain:

- Compiler version
- ABI
- Contract bytecode

But also the documentation generated by Natspec comments
(in the “output” section at the end of the file)

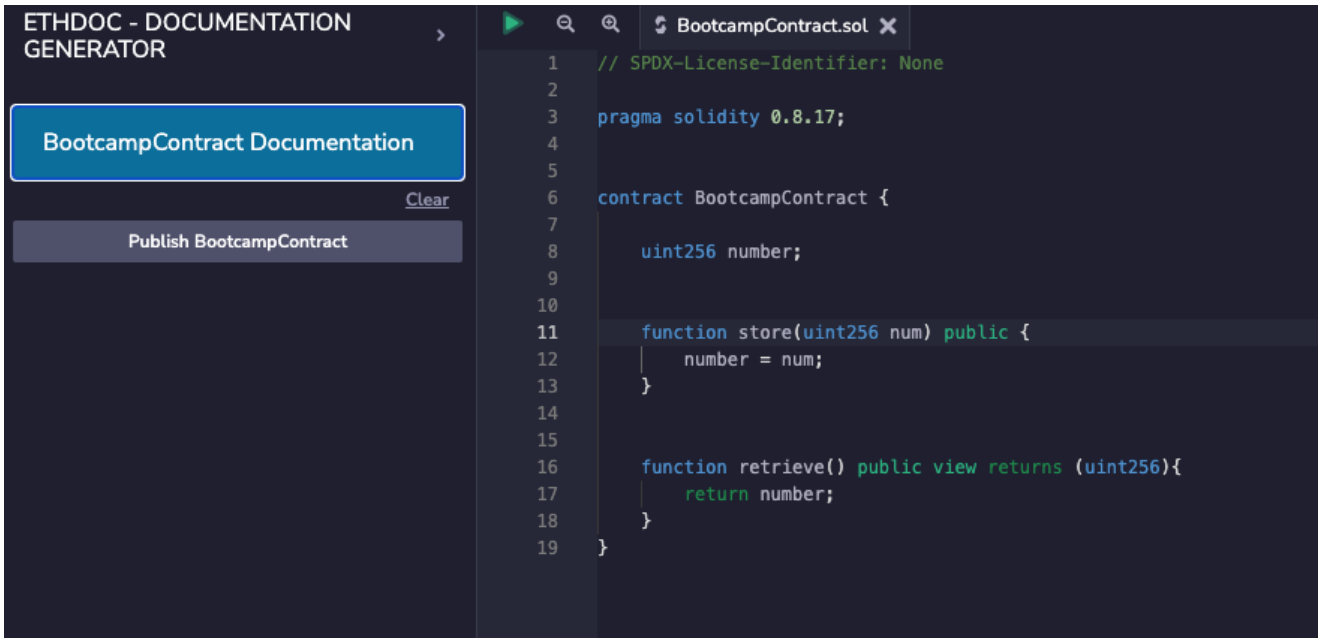
NB: when doing truffle compile, look at the JSON file under the /build folder. If your contract had Natspec comments in it, you will see the “devdoc” and “userdoc” sections.

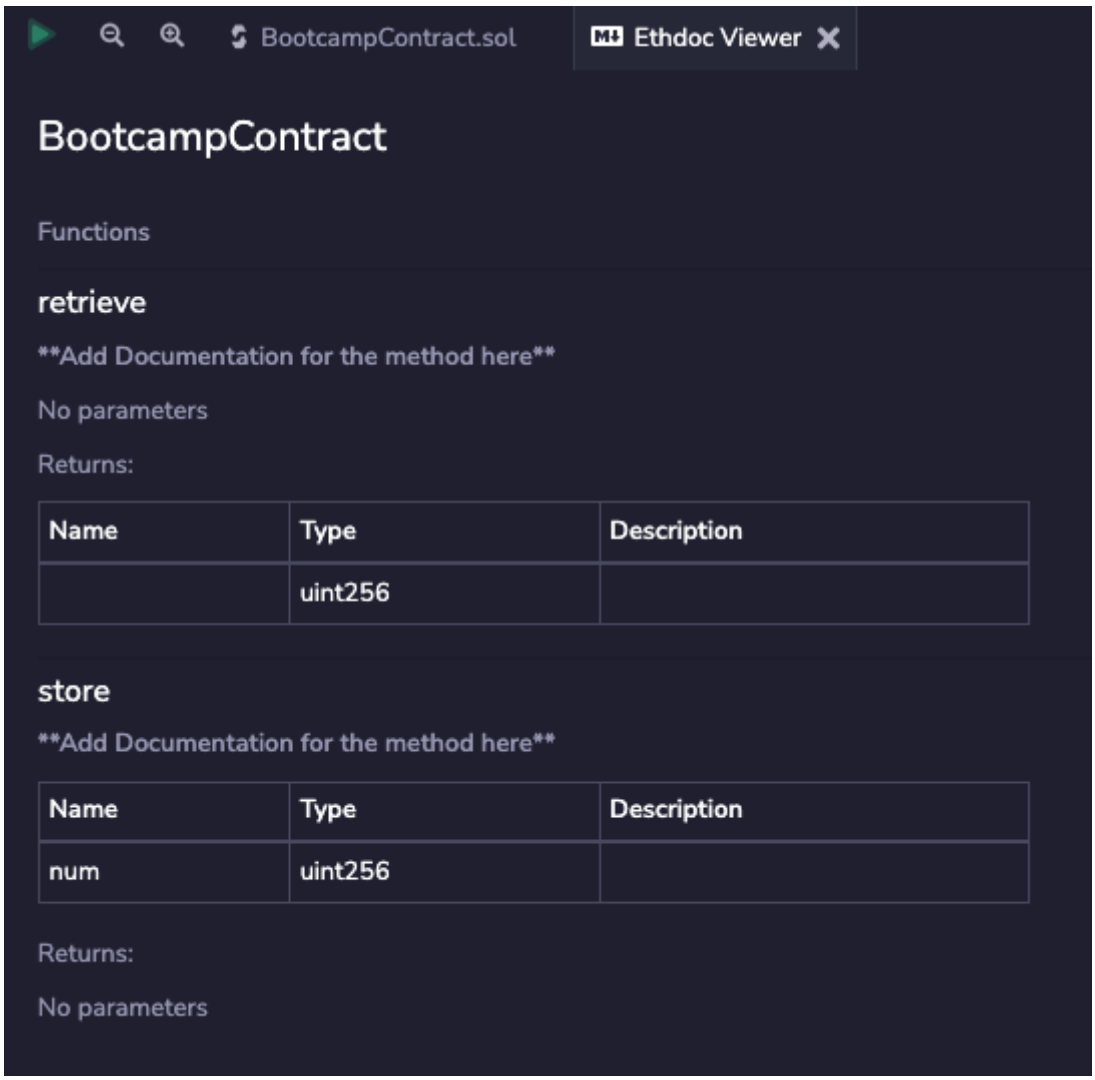
Developer Docs output

Function signature
= function name + parameter types in parentheses (without spaces)
= hashing this with keccak256 gives you the bytes4 function selector

Only public and external functions are shown
(private and internal functions are not parsed)

User Docs output - using Remix EthDoc plugin
NB: at the current time, the EthDoc - Documentation Generator seems to not be working properly. Only the EthDoc viewer is.





Solidity compiler ⇒ examine NatSpec comments ⇒ generate JSON. End user software (eg: Metamask) can consume this document.

Example with @notice tag.

End user call function with `a = 10` as parameter

Parsing @notice tag using Radspec interpreter

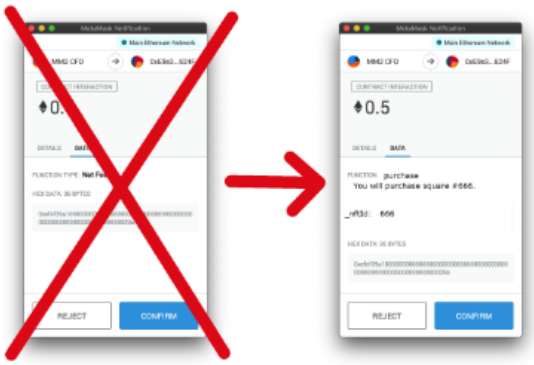
From Aragon

radspec 🤘

build passing coverage repository not found

Radspec is a safe interpreter for dynamic expressions in Ethereum's NatSpec.

This allows smart contact developers to show improved function documentation to end users, without the security pitfalls of natspec.js. Radspec defines its own syntax structure and parses its own AST rather than directly evaluating untrusted JavaScript.



Features

- **Expressive**: Show relevant details to smart contract end-users at the time they make transactions.
- **External calls**: Radspec can query other contracts.
- **Safe**: Radspec requires no DOM access or untrusted JavaScript evaluation.
- **Compatible**: Most existing NatSpec dynamic expressions are compatible with Radspec.

References
https://docs.soliditylang.org/en/latest/natspec-format.html
https://jeancvllr.medium.com/solidity-tutorial-all-about-comments-bc31c729975a
https://www.bitdegree.org/learn/solidity-syntax#natspec
https://github.com/aragon/radspec