Solidity Domain for Sphinx Documentation

Release 0.5.2

Alan Lu

Dec 16, 2020

Contents:

CHAPTER 1

Installation

This package provides a Solidity domain for Sphinx, as well as source autodocumenting functionality. Install this package via pip:

```
pip install sphinxcontrib-soliditydomain
```

and add it to your Sphinx configuration file:

```
extensions = [
    # ...,
    'sphinxcontrib.soliditydomain',
]
```

If the `autodoc` features of this package are desired, be sure that the `sphinx.ext.autodoc` extension is also enabled:

```
extensions = [
    # ...,
    'sphinx.ext.autodoc',
    'sphinxcontrib.soliditydomain',
]
```

Warning: This package was written in Python 3 and **will not** work in Python 2.

If you are using Read the Docs, be sure to set the *Python interpreter* used for the project to the CPython 3.x interpreter. This setting may be found in the *Admin* > *Advanced Settings* menu.

CHAPTER 2

Formatting Solidity Elements

`.. sol:contract:: Name is Parent1, Parent2, ...`
`.. sol:interface:: Name is Parent1, Parent2, ...`
`.. sol:library:: Name`

These directives describe top-level Solidity objects. The following:

```
.. sol:library:: SafeMath

   Provides arithemetic operations guarded against overflow
```

renders as

library **`SafeMath`**
Provides arithemetic operations guarded against overflow

Likewise, the following:

```
.. sol:contract:: MintableBurnableToken is MintableToken, StandardBurnableToken

   A token which can both be minted and burnt.
```

renders as

contract **`MintableBurnableToken`** is MintableToken, StandardBurnableToken
A token which can both be minted and burnt.

`.. sol:statevar:: type visibility name`

State variables in Solidity:

```
.. sol:statevar:: int128 public widgetSocket

   A socket for a widget.
```

yields

int128 *public* **`widgetSocket`**
A socket for a widget.

Visibility modifiers are optional and include the following:

- public
- private
- internal

`.. sol:constructor:: (type mod arg1, type mod arg2, ...) mod1 mod2 ...`

Constructors for contracts. May be used in the context of a *sol:contract* directive. For example,

```
.. sol:contract:: FooFactory

    Produces instances of Foo.

    .. sol:constructor:: (uint a, int b, bytes32 c) public restrictedTo(a)

        Creates a FooFactory, initializing with supplied parameters.
```

yields

contract **FooFactory**

Produces instances of Foo.

constructor (*uint a*, *int b*, *bytes32 c*) *public*

Creates a FooFactory, initializing the new instance with supplied parameters.

`.. sol:function:: name(type mod arg1, ...) mod1 ... returns (type r1, ...)`

Solidity functions. May be used in the context of a *sol:contract*, *sol:library*, or *sol:interface* directive. For example,

```
.. sol:interface:: ERC20

    .. sol:function:: balanceOf(address tokenOwner) \
        public constant returns (uint balance)

    Get the token balance for account ``tokenOwner``.
```

yields

interface **ERC20**

function **balanceOf** (*address tokenOwner*) *public*

Get the token balance for account `tokenOwner`.

`.. sol:modifier:: name(type mod arg1, ...)`

Solidity function modifiers. For example:

```
.. sol:contract:: Ownable

    .. sol:modifier:: onlyOwner()

        Throws if called by any account other than the owner.
```

yields

contract **Ownable**

modifier **onlyOwner**()
Throws if called by any account other than the owner.

.. sol:event:: `name(type mod arg1, ...)`
Solidity events. For example:

```
.. sol:contract:: RefundVault is Ownable

   .. sol:event:: Refunded(address indexed beneficiary, uint256 weiAmount)

      Emitted when ``weiAmount`` gets refunded to a ``beneficiary``.
```

yields

contract **RefundVault** is Ownable

event **Refunded**(*address indexed beneficiary*, *uint256 weiAmount*)
Emitted when `weiAmount` gets refunded to a `beneficiary`.

.. sol:struct:: `Name`
Solidity structs. Members of the struct are represented by a `member` field. For example:

```
.. sol:struct:: DreamMachine

   Some archetypical madness.

   :member uint widget: Funky lil' widget.
   :member FunkUtils.Orientation orientation: Which way the machine is pointing.
   :member typelessThing: Type information is optional.
```

yields

struct **DreamMachine**
Some archetypical madness.

Members

- **widget** (*uint*) – Funky lil' widget.
- **orientation** (*FunkUtils.Orientation*) – Which way the machine is pointing.
- **typelessThing** – Type information is optional.

.. sol:enum:: `Name`
Solidity enum definitions. Like `struct`, members are represented by a `member` field, but for enums, this field is typeless. For example:

```
.. sol:enum:: Direction

   Which way to go.

   :member North: Where Santa's at.
   :member South: Where penguins're at.
   :member East: Get tricky.
   :member West: Get funky.
```

yields

enum **Direction**
Which way to go.

Members

- **`North`** – Where Santa's at.
- **`South`** – Where penguins're at.
- **`East`** – Get tricky.
- **`West`** – Get funky.

CHAPTER 3

Automatic Documentation Generation from Solidity Source

3.1 Configuration

By default, `sphinxcontrib.soliditydomain` assumes that associated Solidity source files may be found in the directory `../contracts` relative to the root of the Sphinx project:

```
.               # project root
├── docs/       # root of the Sphinx project
├── contracts/  # root of all contracts
└── ...
```

This may be changed with the following configuration variable:

`autodoc_lookup_path`
A path to Solidity files to be indexed for autodocumentation purposes. By default, this is `../contracts` relative to the documentation directory.

Note: `sphinxcontrib.soliditydomain` will crawl the contract lookup directory, collecting `.sol` files, parsing the source content with an ANTLR 4 parser using this Solidity grammar definition, and building a database of Solidity language objects for which the documentation tool will be able to automatically generate documentation.

Note: If a Solidity source file cannot be parsed by this package, a warning will be issued and the Sphinx build will continue trying to build the rest of the documentation.

3.2 Autodoc Directives By Example

All of the formatting directives admit corresponding autodocumentation directives accessible by prepending `autosol` to the formatting directive name.

Let's suppose that the following code is found in a Solidity source file:

```
pragma solidity ^0.4.24;

library DaHood {
    enum Coast { East, West }
}

/// @title A simulator for Bug Bunny, the most famous Rabbit
/// @author Warned Bros
contract BugBunny {

    /// Hash of a carrot. You can use triple forward slashes (``///``)
    /// to have Solidity Domain pull the docs out of the comment.
    /**
     * Comment blocks starting with ``/**`` will also be added to documentation.
     * These blocks may be framed with a preceding ``*`` on each line.
     */
    bytes32 public carrotHash;
    mapping (address => mapping (uint => bool)) public ballerz;

    event Consumption(address indexed feeder, string food);
    event Consumption(address indexed payer, uint amount);

    /// Doxygen-style tags on events currently unsupported by devdocs
    /// but will work here.
    /// @param coast The original beef.
    event AnonEvent(DaHood.Coast coast) anonymous;

    /// Constructor for BugBunny. Note that solc doesn't parse
    /// Doxygen-style devdocs for these, but this is supported
    /// in this plugin.
    /// @param carrot Eh... what's up, doc?
    constructor(string carrot) public {
        carrotHash = keccak256(abi.encodePacked(carrot));
    }

    /// @author Birb Lampkett
    /// @notice Determine if Bug will accept `_food` to eat
    /// @dev String comparison may be inefficient
    /// @param _food The name of a food to evaluate (English)
    /// @return true if Bug will eat it, false otherwise
    function doesEat(string _food) public view returns (bool) {
        return keccak256(abi.encodePacked(_food)) == carrotHash;
    }

    /// @author Funk Master
    /// @dev Magic funk machine wow.
    /// @param _food The name of a food to eat
    /// @return {
    ///     "eaten": "true if Bug will eat it, false otherwise",
    ///     "hash": "hash of the food to eat"
    /// }
    function eat(string _food) public returns (bool eaten, bytes32 hash) {
        eaten = doesEat(_food);
        hash = keccak256(abi.encodePacked(_food));
        if(eaten) {
            emit Consumption(msg.sender, _food);
```

(continues on next page)

(continued from previous page)

```
        }
    }

    /// @notice Bug will eat either `food1` or `food2`
    /// @dev Raw stuff.
    /// @param food1 The name of first food to try
    /// @param food2 The name of second food to try
    /// @return {
    ///    "eaten": "true if Bug ate, false otherwise",
    ///    "hash": "hash of the food eaten"
    /// }
    function eat(string food1, string food2) external returns (bool eaten, bytes32␣
↪hash) {
        if(doesEat(food1)) {
            (eaten, hash) = eat(food1);
        } else {
            (eaten, hash) = eat(food2);
        }
    }

    // tags on fallback functions currently not supported by devdocs
    function() external payable {
        emit Consumption(msg.sender, msg.value);
        ballerz[msg.sender][msg.value] = true;
    }
}
```

The following directives may be used:

.. autosolcontract:: Name
.. autosollibrary:: Name
.. autosolinterface:: Name

These directive require the targetted object's name and will render to a corresponding `sol:contract`, `sol:library`, or `sol:interface` block. The following ReST:

```
.. autosolcontract:: BugBunny
```

will render like so:

contract **BugBunny**

Title A simulator for Bug Bunny, the most famous Rabbit

Author Warned Bros

Furthermore, the `:noindex:`, `:members:` and `:exclude-members:` options may be used as expected, with

```
.. autosolcontract:: BugBunny
    :noindex:
    :members: doesEat, constructor
```

yielding

contract **BugBunny**

Title A simulator for Bug Bunny, the most famous Rabbit

Author Warned Bros

constructor (*string carrot*) *public*

Constructor for BugBunny. Note that solc doesn't parse Doxygen-style devdocs for these, but this is supported in this plugin.

Parameters

- **carrot** – Eh… what's up, doc?

function **doesEat** (*string _food*) *public*

String comparison may be inefficient

Author Birb Lampkett

Notice Determine if Bug will accept *_food* to eat

Parameters

- **_food** – The name of a food to evaluate (English)

Return true if Bug will eat it, false otherwise

and

```
.. autosolcontract:: BugBunny
   :noindex:
   :members:
   :exclude-members: ballerz, Consumption, eat, doesEat, <fallback>
```

yielding

contract **BugBunny**

Title A simulator for Bug Bunny, the most famous Rabbit

Author Warned Bros

bytes32 *public* **carrotHash**

Hash of a carrot. You can use triple forward slashes (`///`) to have Solidity Domain pull the docs out of the comment.

Comment blocks starting with `/**` will also be added to documentation. These blocks may be framed with a preceding `*` on each line.

event **AnonEvent** (*DaHood.Coast coast*) *anonymous*

Doxygen-style tags on events currently unsupported by devdocs but will work here.

Parameters

- **coast** – The original beef.

constructor (*string carrot*) *public*

Constructor for BugBunny. Note that solc doesn't parse Doxygen-style devdocs for these, but this is supported in this plugin.

Parameters

- **carrot** – Eh… what's up, doc?

Note: Contract members will appear in the order they were indexed by the Solidity source crawler.

CHAPTER 4

Cross Referencing Solidity Objects

:sol:contract:
:sol:lib:
:sol:interface:
:sol:svar:
:sol:cons:
:sol:func:
:sol:mod:
:sol:event:
:sol:struct:
:sol:enum:

These roles aid in cross referencing Solidity objects in the same project. For example,

```
:sol:func:`balanceOf`
```

will render as *balanceOf*, which will link to where in the documentation this function has been documented. Likewise, autodoc generated documentation can be cross-referenced as well. For example,

```
:sol:contract:`BugBunny`
```

will refer to the *BugBunny* documentation which has been indexed.

Using the `:noindex:` option will prevent a Solidity object description from being cross-referenced.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index