LaTeX Highlighting for Solidity and Yul

Sam Bacha[1]

Manifold Finance, Inc
{sam}@manifoldfinance.com

1 Usage

Uses the python package, pygments-lexer-solidity package

2 Solidity

2.1 example.sol - Solidity

```
// SPDX-License-Identifier: BSD-2-Clause

pragma solidity ^0.6.0;
pragma ABIEncoderV2;
pragma experimental SMTChecker;
/**************************************************************************
 *                            example.sol                                 *
 **************************************************************************/

// Code in this contract is not meant to work (or be a good example).
// It is meant to demonstrate good syntax highlighting by the lexer,
// even if otherwise hazardous.

// Comments relevant to the lexer are single-line.
/* Comments relevant to the code are multi-line. */

library Assembly {
    function junk(address _addr) private returns (address _ret) {
        assembly {
            let tmp := 0

            // nested code block
            let mulmod_ := 0 { // evade collision with `mulmod`
                let tmp:=sub(mulmod_,1) // `tmp` is not a label
                mulmod_ := tmp
            }
            /* guess what mulmod_ is now... */
        _loop: // JIC, dots are invalid in labels
            let i := 0x10
```

```
            loop:
                // Escape sequences in comments are not parsed.
                /* Not sure what's going on here, but it sure is funky!
                 \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ \o/ */
                mulmod(_addr, mulmod_, 160)

                0x1 i sub // instructional style
                i =: tmp /* tmp not used */

                jumpi(loop, not(iszero(i)))

                mstore(0x0, _addr)
                return(0x0, 160)
            }
        }
    }

    contract Strings {
        // `double` is not a keyword (yet)
        string double = "This\ is a string\nwith \"escapes\",\
    and it's multi-line. // no comment"; // comment ok // even nested :)
        string single = 'This\ is a string\nwith "escapes",\
    and it\'s multi-line. // no comment'; // same thing, single-quote
        string hexstr = hex'537472696e67732e73656e6428746869732e62616c616e6365293b';

        fallback() external payable virtual {}

        receive() external payable {
            revert();
        }
    }

    contract Types is Strings {
        using Assembly for Assembly;

        bytes stringsruntime = type(Strings).runtimeCode;

        // typesM (compiler chokes on invalid)
        int8 i8;           // valid
        //int10 i10;       // invalid
        uint256 ui256;     // valid
        //uint9001 ui9001; // invalid
        bytes1 b1;         //valid
        //bytes42 b42;     // invalid - M out of range for `bytes`
```

```
// typesMxN (compiler chokes on invalid)
fixed8x0 f8x0;           // valid
fixed8x1 f8x1;           // valid
fixed8x8 f8x8;           // valid
//fixed0x8 f0x8;         // invalid since MxN scheme changed
ufixed256x80 uf256x80;   // valid
//ufixed42x217 uf42x217; // invalid - M must be multiple of 8, N <= 80

// special cases (internally not types)
string str; // dynamic array (not a value-type)
bytes bs; // same as above
//var v = 5; // `var` is a keyword, not a type, and compiler chokes
uint unu$ed; // `var` is highlighted, though, and `$` is a valid char

address a = "0x1"; // lexer parses as string
struct AddressMap {
    address origin;
    address result;
    address sender;
    bool touched;
}
mapping (address => AddressMap) touchedMe;

function failOnNegative(int8 _arg)
    private
    pure
    returns (uint256)
{
    /* implicit type conversion from `int8` to `uint256` */
    return _arg;
}

// some arithmetic operators + built-in names
function opportunisticSend(address k) private {
    /* `touchedMe[k].result` et al are addresses, so
       `send()` available */
    touchedMe[k].origin.send(uint256(k)**2 % 100 finney);
    touchedMe[k].result.send(1 wei);
    touchedMe[k].sender.send(mulmod(1 szabo, k, 42));
}

fallback() external payable override {
    /* inferred type: address */
    var k = msg.sender;
    /* inferred type: `ufixed0x256` */
```

```
        var v = 1/42;
        /* can't be `var` - location specifier requires explicit type */
        int negative = -1;

        // valid syntax, unexpected result - not our problem
        ui256 = failOnNegative(negative);

        // logic operators
        if ((!touchedMe[msg.sender].touched &&
             !touchedMe[tx.origin].touched) ||
            ((~(msg.sender * v + a)) % 256 == 42)
        ) {
            address garbled = Assembly.junk(a + msg.sender);

            /* create a new AddressMap struct in storage */
            AddressMap storage tmp;

            // TODO: highlight all known internal keywords?
            tmp.origin = tx.origin;
            tmp.result = garbled;
            tmp.sender = msg.sender;
            tmp.touched = true;

            /* does this link-by-reference as expected?.. */
            touchedMe[msg.sender] = tmp;
            touchedMe[tx.origin] = tmp;
        }
        else {
            /* weak guard against re-entry */
            touchedMe[k].touched = false;

            opportunisticSend(k);

            delete touchedMe[k];
            /* these probably do nothing... */
            delete touchedMe[msg.sender];
            delete touchedMe[tx.origin];
        }
    }
}

/**
   \brief Examples of bad practices.

   TODO: This special NatSpec notation is not parsed yet.
```

```
   @author Noel Maersk
 */
/// Triple-slash NatSpec should work.
/// @title Some examples of bad practices.
/// @author Noel Maersk
/// @notice Very old, might've been "fixed" by obsoletion.
/// @dev This is a dummy comment.
/// @custom:unmaintained This code is not maintained.

contract BadPractices {
    address constant creator; /* `internal` by default */
    address private owner; /* forbid inheritance */
    bool mutex;

    modifier critical {
        assert(!mutex);
        mutex = true;
        _;
        mutex = false;
    }

    constructor() external {
        creator = tx.origin;
        owner = msg.sender;
    }

    /* Dangerous - function public, and doesn't check who's calling. */
    function withdraw(uint _amount)
        public
        critical
        returns (bool)
    { /* `mutex` set via modifier */
        /* Throwing on failed call may be dangerous. Consider
           returning false instead?.. */
        require(msg.sender.call.value(_amount)());
        return true;
    } /* `mutex` reset via modifier */

    /* fallback */
    fallback() external payable {
        /* `i` will be `uint8`, so this is an endless loop
           that will consume all gas and eventually throw.
         */
        for (var i = 0; i < 257; i++) {
```

```
            owner++;
        }
    }

    /* receive()?.. nah, why bother */
}

/* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /* /*
// A regular multi-line comment closure, including an escaped variant as
// demonstrated shortly, should close the comment; note that the lexer
// should not be nesting multi-line comments.
//
// If the comment is still shown as "open", then a
//
//                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//                !!! MALICIOUS CODE SEGMENT !!!
//                !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//
// can be erroneously thought of as inactive, and left unread.
// In fact, the compiler will produce executable code it, possibly
// overriding the program above.
//
// It is imperative that syntax highlighters do parse it if either of
// `* /` or `\* /` (with space removed) are present.
//
// Now, let's party! :) \*/

contract MoreBadPractices is BadPractices {
    uint balance;

    fallback() external payable override {
        balance += msg.value;
        if (!msg.sender.send(this.balance / 10)) throw;
        balance -= this.balance;
    }
}

/*
// Open comment to EOF. Compiler chokes on this, but it's useful for
// highlighting to show that there's an unmatched multi-line comment
// open.

contract CommentToEndOfFile is MoreBadPractices {
    fallback() external payable override {}
}
```

2.2 example.yul - Yul

```
{
    // my function
    function power(base, exponent) -> result
    {
        switch exponent
        case 0 { result := 1 }
        case 1 { result := base }
        default
        {
            result := power(mul(base, base), div(exponent, 2))
            switch mod(exponent, 2)
                case 1 { result := mul(base, result) }
        }
    }
}
```

1

[1] Taken from example.yul

This document presents several examples showing how to use the `pygments-lexar-solidity` and `xcolor` package to provide syntax highlighting for the programming languages Solidity and Yul as well as changing the colour of LaTeX page elements.

- First item
- Second item