ethereum vienna

Workshop
From Idea to Contract

Workshops

Workshop #1: Contract Development for Beginners

Requirements: Basic Understanding of Ethereum

Solidity Basics

Workshop #2: From Idea to Contract

Requirements: Basic Understanding of Solidity

Mapping the real world to ethereum concepts

Advanced Solidity

Workshop #3: From Contract to DApp

Requirements: Basic Understanding of Solidity, HTML/JS, node.js

Interfacing with Ethereum using web3.js

Auxiliary Technologies: IPFS, Whisper and Swarm

Agenda

1. Mapping real world concepts to solidity
2. “Advanced” Solidity
3. Standard Contracts
4. **Developing the Contract**
5. Testing
6. **Testing the Contract**
7. Deployment

Warnings

This workshop does **not** make you a contract developer

many small but important differences to other languages

=> many possible bugs (stuck contract, stolen funds, etc.)

If you ever intend to make a real world contract

read the solidity documentation **in its entirety**

tests ,tests, tests

Contracts

Only a image of the state of the real world

Change in RW not reflected in contract

Change in contract not reflecting change in RW

Only on-blockchain data is fully secured

ether, tokens, owners of blockchain assets

Real World -> Contracts

RW Actors represented by (external) accounts. (Not sybil safe)

RW Information cannot be trusted in a trustless way

Many RW Applications can never be fully trustless

=> minimise required trust (multisig, incentives, etc.)

Many "trustless" applications still have trusted points

(e.g. amazon in every oraclize call, flightstats in etherisc)

Real World -> Contracts

Mapping

When something has a non-incrementing identifier

All Fields 0 is either

the default state for some key

or an invalid state

cannot be enumerated

Ideal when you need to map from addresses to something

Real World -> Contracts

Array

When the index can be derived from the id

Incrementing id

When you need to enumerate a collection

Sometimes an array and a mapping to the same data can be useful

Real World -> Contracts

Events

Everything something in the RW needs to react to

Only way to be light client friendly

If something only needs to be enumerated in RW

events can replace array (no storage cost)

RW actor iterates over events instead

Real World -> Contracts

External Data

Trusted Data Feed

Multisig

3rd party with minimised trust (like oraclize for web resources)

Cryptlets (is this a thing already?)

Solidity

Solidity Contract Interaction

"Interface"

```
contract token {
    mapping (address => uint) public coinBalanceOf;
    function token() {}
    function sendCoin(address receiver, uint amount) returns(bool sufficient) {  }
}
```

Contains just the function headers

Gives the compiler with the necessary info for calls

Solidity Contract Interaction

Usage

```
function test() returns (uint) {
    address v = 0x1338924 7f732748971389738714738974 9823479;
    /* cast address to contract type */
    token t = token(v);
    /* call sendCoin function */
    t.sendCoin(0xffffffffffffffffffffffffffffffffffffffff, 1);
    /* call sendCoin function with 10000 gas limit and send one ether */
    t.sendCoin.gas(10000).value(1 ether)(0xffffffffffffffffffffffffffffffffffffffff, 1);
    /* calling autogenerated getter */
    return t.coinBalanceOf(0xffffffffffffffffffffffffffffffffffffffff);
}
```

Call Types

send	sends a message with 0 gas (stipend only)
call	sends a message with gas
callcode	call code in the current context
delegatecall	like callcode, but preserves sender and value
create	creates contract and runs init code

Solidity Contract Interaction

Creation

```
contract a { }

contract b {
    a v;

    function b () {
        v = new a();
    }
}
```

Solidity Import

```
import "filename";
import * as symbolName from "filename";
import {symbol1 as alias, symbol2} from "filename";
import "filename" as symbolName;
```

Solidity Libraries

Used to share code between contracts

```
library Set {
  // We define a new struct datatype that will be used to
  // hold its data in the calling contract.
  struct Data { mapping(uint => bool) flags; }

  // Note that the first parameter is of type "storage
  // reference" and thus only its storage address and not
  // its contents is passed as part of the call.  This is a
  // special feature of library functions.  It is idiomatic
  // to call the first parameter 'self', if the function can
  // be seen as a method of that object.
  function insert(Data storage self, uint value)
      returns (bool)
  {
      if (self.flags[value])
          return false; // already there
      self.flags[value] = true;
      return true;
  }
}
```

```
contract C {
    Set.Data knownValues;

    function register(uint value) {
        // The library functions can be called without a
        // specific instance of the library, since the
        // "instance" will be the current contract.
        if (!Set.insert(knownValues, value))
            throw;
    }
}
```

Solidity Contract Inheritance

```
contract owned {
    function owned() { owner = msg.sender; }
    address owner;
}

// Use "is" to derive from another contract. Derived
// contracts can access all non-private members including
// internal functions and state variables. These cannot be
// accessed externally via `this`, though.
contract mortal is owned {
    function kill() {
        if (msg.sender == owner) selfdestruct(owner);
    }
}
```

Solidity Contract Inheritance

Abstract Contract

```
contract Config {
    function lookup(uint id) returns (address adr);
}
```

Multiple Inheritance

```
contract named is owned, mortal {
    function named(bytes32 name) {
    }
    // Functions can be overridden, both local and
    // message-based function calls take these overrides
    // into account.
    function kill() {
        super.kill();
    }
}
```

Solidity Contract Inheritance

Passing constructor arguments

```
contract Base {
    uint x;
    function Base(uint _x) { x = _x; }
}

contract Derived1 is Base(7) {
    function Derived1() {
    }
}

contract Derived2 is Base {
    function Derived2(uint _y) Base(_y * _y) {
    }
}
```

Standard Contracts

import "std" (only works with solc and --add-std)

```
pragma solidity >= 0.4.5;
import "std.sol";

contract Attendance is owned {
  struct Attendee {
    bytes32 name;
    mapping (uint => bool) attended;
  }

  mapping (bytes32 => address) reverseMap;
  mapping (address => Attendee) public attendees;
  bytes32[] public events;
```

Standard Contract: Token

```
contract Token {
    function totalSupply() constant returns (uint256 supply) {}
    function balanceOf(address _owner) constant returns (uint256 balance) {}
    function transfer(address _to, uint256 _value) returns (bool success) {}
    function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {}
    function approve(address _spender, uint256 _value) returns (bool success) {}
    function allowance(address _owner, address _spender) constant returns (uint256 remaining) {}

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

Standard Contracts

token registry (only EIP)

ENS: Ethereum Name System (now on testnet)

Reverse ENS has been proposed

Only real standard today is the token standard

Getting contracts into geth

Use web3.js

Ready JS available from the solidity online compiler

web3.js javascript object represents a contract

let Contract = eth.contract(abiGoesHere)

let instance = contract.at(address)

instance.method(arguments, txinfo, callback)

Default ideas

Marketplace + Reputation + Trusted Identity Guard

Vickrey Auction

Word Guessing Game

Token Market

Multi Subscription Contract (Collect all payments)

Your turn:

Developing the contract

Testing

Mostly done in Solidity or/and JS

Many different approaches

All of them come with disadvantages

Testing with Solidity

Dapple

Runs in its own vm environment

Tests are solidity contracts

Contracts to represent identities

Solidity + assertions

drawback: fixed block and tx data across all tests

Testing with JS

Embark

Complete development framework

Comes with custom libraries

Has its own test library

Also abstracts away things like IPFS

drawback: very opinionated

Testing with JS

Truffle

Complete development framework

Pudding library (optional)

Uses mocha / chai

drawback: still quite opinionated

Testing - Other

populus / py.test

JAVA and ethereumJ

ethereumjs-testrpc

node + ethereumjs-vm

Test Scenarios

All use cases

Unauthorised access

Users sends too few / too much

Test Scenarios

Different gas limits (to detect uncaught errors)

```
contract token {
    mapping (address => uint) balances;
    function send (address a, uint256 v) {
        if(balances[a] >= v) {
            balances[a] -= v;
            a.send(v);
        }
    }
}
```

Solidity compiler will give a warning for this now!

Test Scenarios

Different gas limits (to detect uncaught errors)

Only really necessary if you use explicit gas limits and want to assume it works

Gas mechanics quite complex:

send cost varies based on account existence

Test Scenarios

Call Recursion / Reentrance

```
contract token {
    mapping (address => uint) balances;
    function send (address a, uint256 v) {
        if(balances[a] >= v) {
            if(a.call.value(v)()) {
                balances[a] -= v;
            }
        }
    }
}
```

Test Scenarios

Contract Reentrance

During **any untrusted call** you contract can be called again (unless low gaslimit)

After **any untrusted call**, all assumptions about the state of the contract are invalid

=> If you use a prevention, make sure it actually works!

Testing: This Workshop

Both solidity and JS testing:

Solidity with dapple

JS with mocca / chaithereum and web3-q

- chaithereum: chai plugin for ethereum
- web3-q: promisified web3 library

Testing: This Workshop

Demo

Your turn:

Testing the contract

Deployment

UI Wallet

geth

node.js

Ethereum Studio

Frameworks with their own deployment mechanism

Deployment using UI

Demo

Deployment using Geth

Demo

Deployment using node.js

Demo

1vieCmqYB3DE8StinXYBGGvgJ9hoXP1ib

The End

0x50008dd0cc879e0341042f97541eb4870c9c8393