Ethereum Smart Contracts

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

September 4, 2018

Ethereum Contracts

- Contract = Collection of functions and state at a specific address
  - Account state = [nonce, balance, storageRoot, codeHash]
- Created by contract creation transactions
- Contract logic is stored in EVM bytecode
- Written in a high level language which compiles to bytecode
  - Solidity `https://solidity.readthedocs.io`
  - Vyper `https://vyper.readthedocs.io`
- Anatomy of a contract
  - State variables
  - Functions
  - Events

Currency Example

Currency Example

```
1  pragma solidity ^0.4.7;
2
3  contract Coin {
4      address public minter;
5      mapping (address => uint) public balances;
6
7      event Sent(address from, address to, uint amount);
8
9      constructor() public {
10         minter = msg.sender;
11     }
12
13     function mint(address receiver, uint amount) public {
14         if (msg.sender != minter) return;
15         balances[receiver] += amount;
16     }
17
18     function send(address receiver, uint amount) public {
19         if (balances[msg.sender] < amount) return;
20         balances[msg.sender] -= amount;
21         balances[receiver] += amount;
22         emit Sent(msg.sender, receiver, amount);
23     }
24 }
```

Currency Example Anatomy

- State variables

```
address public minter;
mapping (address => uint) public balances;
```

- Functions

```
constructor() public {..}

function mint(address receiver, uint amount) public {..}

function send(address receiver, uint amount) public {..}
```

- Events

```
event Sent(address from, address to, uint amount);
```

Contract Creation and Currency Allotment

- At contract creation, `minter` is initialized to creator

```
address public minter;

constructor() public {
    minter = msg.sender;
}
```

- `minter` can call `mint` and allot currency to addresses

```
mapping (address => uint) public balances;

function mint(address receiver, uint amount) public {
    if (msg.sender != minter) return;
    balances[receiver] += amount;
}
```

- Public functions form the contract interface (can be called via message call)
- Private functions and variables are only visible in original contract, not in derived contracts

Currency Transfers

```
mapping (address => uint) public balances;

event Sent(address from, address to, uint amount);

function send(address receiver, uint amount) public {
    if (balances[msg.sender] < amount) return;
    balances[msg.sender] -= amount;
    balances[receiver] += amount;
    emit Sent(msg.sender, receiver, amount);
}
```

- Once allotted currency, address owners can transfer to others
- An event is emitted to enable light clients to find this log
- Remix Demo `https://remix.ethereum.org`

Open Auction

Open Auction

- Bids are known to everyone
- State variables

```
// Address of auction beneficiary
address public beneficiary;

// Auction end time in Unix time
uint public auctionEndTime;

// Current state of the auction.
address public highestBidder;
uint public highestBid;

// Allowed withdrawals of previous bids
mapping(address => uint) pendingReturns;

// Set to true at the end, disallows any change
bool ended;
```

- Events

```
event HighestBidIncreased(address bidder, uint amount);
event AuctionEnded(address winner, uint amount);
```

Contract Creation

```
constructor(
    uint _biddingTime,
    address _beneficiary
) public {
    beneficiary = _beneficiary;
    auctionEndTime = now + _biddingTime;
}
```

- Initialize `beneficiary` and `auctionEndTime`
- Contract creation transaction will take arguments as inputs

Making a Bid

```
function bid() public payable {

  require(now <= auctionEndTime, "Auction already ended.");

  require(msg.value > highestBid, "There already is a
      higher bid.");

  if (highestBid != 0) {
      pendingReturns[highestBidder] += highestBid;
  }
  highestBidder = msg.sender;
  highestBid = msg.value;
  emit HighestBidIncreased(msg.sender, msg.value);
}
```

- `payable` keyword allows Ether to be sent with message call
- Check that auction is ongoing and new bid is highest bid
- If new bid is higher, add old highest bid to `pendingReturns` list
- Emit event notifying change in highest bid

Withdraw Losing Bids

```
function withdraw() public returns (bool) {
    uint amount = pendingReturns[msg.sender];
    if (amount > 0) {
        pendingReturns[msg.sender] = 0;

        if (!msg.sender.send(amount)) {
            pendingReturns[msg.sender] = amount;
            return false;
        }
    }
    return true;
}
```

- Set balance of withdrawer to zero
- If withdrawal fails, restore `amount` in `pendingReturns` and return `false`
- If withdrawal succeeds, return `true`

Ending the Auction

```
function auctionEnd() public {

  require(now >= auctionEndTime, "Auction not yet ended.");
  require(!ended, "auctionEnd has already been called.");

  ended = true;
  emit AuctionEnded(highestBidder, highestBid);

  beneficiary.transfer(highestBid);
}
```

- Check that `auctionEndTime` has passed
- Check that `auctionEnd` has not been called before
- Emit event signaling end of auction
- Transfer highest bid to beneficiary

References

- Solidity Documentation `https://solidity.readthedocs.io`
- Remix IDE `https://remix.ethereum.org`