Writing a Contract That Handles Time

JANUARY 12, 2018 BY TODD PROEBSTING

This post will demonstrate how to write a simple, but complete, smart contract in Solidity that deals with time. It assumes that you have read our previous post, Checking the Sender In a Smart Contract.

Sometimes you may want your smart contract to respond to transactions differently after some point in the future. To do this, the contract needs to have a way to express and store values that represent time. The Ethereum Virtual Machine represents time as the (integer) number of seconds since the "Unix epoch", and the current time is accessible to a Solidity program as `now`, which is an alias for `block.timestamp`.

The code below creates a smart contract that does nothing beyond remembering when it was created:

```
pragma solidity ^0.4.19;

contract Time {
    uint256 public createTime;

    function Time() public {
        createTime = now;
    }
}
```

This contract demonstrates the following:

- The system-defined variable `now` contains the current time in seconds since the Unix epoch. This is defined as the timestamp of the block (on the blockchain) that the transaction is a part of.
- The natural unit of time in the EVM is seconds.

Relative Time

To demonstrate using time to influence transaction processing, I am going to develop a smart contract to help me save money for a specified period of time. During that period, I (or anybody else!) can deposit ether in the contract, but the contract will not let me withdraw any ether until the waiting period is over.

Solidity has "time units" built in. I can specify `3 days` or `5 years`, for instance. The savings contract will be parameterized by a waiting period specified in days. Because the EVM naturally stores time in seconds, the contract must scale the waiting period by the number of seconds in a day, which is given by the literal `1 days` in Solidity.

savings.sol

```
pragma solidity ^0.4.19;

contract Savings {
    address owner;
    uint256 deadline;

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);
    }

    function Savings(uint256 numberOfDays) public payable {
        owner = msg.sender;
        deadline = now + (numberOfDays * 1 days);
```

```
        }

        function withdraw() public onlyOwner {
            require(now >= deadline);

            msg.sender.transfer(address(this).balance);
        }
    }
```

The code above demonstrates the following new concepts:

- `(numberOfDays * 1 days)` computes the time in seconds in `numberOfDays` days.
- `now + (numberOfDays * 1 days)` computes the EVM time that represents `numberOfDays` days in the future.
- `now >= deadline` tests if the current time is greater than or equal to `deadline`. (I.e, has the deadline passed?)

The Savings contract is parameterized by the number of days during which no withdrawals are permitted. During that period, any `withdraw` transactions will simply fail. After that time has elapsed, withdrawals will succeed.

Deposits are allowed at any time. Note also that the constructor has the `payable` modifier, which allows ether to be attached (and transferred) to the contract during deployment.

Summary

- Each block in the blockchain includes a timestamp specified as the number of seconds since the Unix epoch. Time values are integers.
- Solidity smart contracts can access the timestamp of the current block as `now` or `block.timestamp`.
- Solidity provides convenient time units like `days` and `years`, which are helpful in computing time spans.

Additional Resources

- Units and Globally Available Variables has more information about units of time in Solidity.

← How Smart Contract Deployment Works

Verifying Contract Source Code →