Leasing AGB

Kestenholz Cars AG

December 3, 2021

Abstract

https://github.com/fridolinvii/Smart_Contract_Leasing_Contract_Example

1 Example Description

This is a PDF example. Hereby we explain the code and the smart contract. This is converted in a unique hash function. We can upload it with the help of ipfs.

Changing the PDF, we get a new hash function. This will help us because the PDF can not be changed during the running of the contract.

2 Code Explanation of Leasing Agreement

2.1 Constructor

The following boundary conditions of the contract are defined through the constructor.

- recipient: This is the beneficiary's wallet address, which receives the prepayment and monthly payments.
- currentOwner: Defines the current owner of the vehicle.
- LeasingTotalCost: The total monetary amount that is owned to the lessor (Kestenholz Cars AG) by the lessee for this leasing contract. Includes both prepayment and monthly payments.
- recurringPayment: Monthly payments.
- VehicleIdentificationNumber: Unique code that serves as an identification of motor vehicles.
- ipfs address of the contract (Leasing AGB + code explenation): Generates redirection via website address from Solidity code to this PDF.

2.2 signContract

The contract can only be signed in case it is not terminated or already signed. Additionally, the prepayment is not allowed to exceed the total leasing costs or be below the specified prepayment value.

By signing the contract the lessee becomes the new owner and a block.timestamp is used to document the date of the contract initiation. After initiation the recurring payments set in.

2.3 sendPayment

Payments can only be send in case the contract is signed and not terminated up to now. The sent payment, which is done in ETH, is being accumulated to the previously sent payments. The sum of the sent payment and accumulated payments is not allowed to exceed the total leasing costs.

2.4 showMinimumBalanceRequired

This function displays the minimum amount that needs to be send in the leasing contract. In our simulation this value updates after each minute, but in reality this value would be changing every month.

It is important to notice that the minimum balance required should not be exceeding the send payments, otherwise the lessor (Kestenholz Cars AG) has the right to terminate the contract and receive the current accumulated payments.

The leasing contract can not be terminated as long as the minimum balance required is below the total leasing costs, indicating that the contract is terminated within the agreed time frame.

2.5 end Contract

The contract can be ended/terminated whenever one of the following events (in chronological order) occurs.

- The leasing contract was not signed in the first place.
- The accumulated payment is equal to the total leasing cost. In this case the ownership is being transferred from our car dealership (Kestenholz Cars AG) to the lessee. We receive the full amount of the accumulated payment.
- The accumulated payment is smaller than minimal required payment. This leads to the breach of contract due to insufficient payments by the lessee, therefore the contract is terminated and no ownership transfer occurs. We receive the current amount of the accumulated payment.

Additionally, the application of the endContract Function generates the exact event that occurred as a comment.

3 Vehicle Example

The following vehicle (and its identification number) for our current leasing contract.

Sample Illustrations: Vehicle Identification Number "WPO ZZZ 91 ZDS 102 886" (Illustration 1) and the vehicle of the leasing contract (Illustration 2)

Within individual leasing contracts individual Vehicle Identification Numbers occur, matching a vehicle to its particular number. For every leasing contract the number can be specified in the Solidity code, creating a smart contract for each individual vehicle. This saves time on additional specification requirements within the Solidity code, such as brand, color, motor specifications and certain equipment elements.

Appendix

Smart Contract Code

This is an overview of the complete Solidity code that we use for our current leasing contract.

```
1  //SPDX-License-Identifier: MIT
2
3  /*
4      Date: November 2021
5      Authors: Carlo Seppi, Andrey Shmelev, Lindijan Alijoski
6      Github: https://github.com/fridolinvii/Smart_Contract_Leasing_Contract_Example
7
8      DISCLAIMER OF LIABILITY: The authors assumes or undertakes NO LIABILITY for any
           loss or damage suffered as a result of the use,
9                                misuse or reliance on the information and content on this
                                      website and the code.
10 */
11
12 pragma solidity ^0.8.10;
13
```

```
14 contract LeasingAgreement {
15
16     // Lessee can pay with multiple address
17     address public immutable recipient; // The account receiving the payments (lessor)
18
19     string public currentOwner;  // Current owner (lessor)
20     string public newOwner;  // New owner (lessee)
21     string public oldOwner;  // after owner transfer, that previous owner is still
           known
22
23     uint public leasingTotalCost;  // Total leasing amount
24     uint public singlePayment; // Amount paid by sender (once)
25     uint public recurringPayment; // Amount paid by sender (monthly)
26
27     uint public createdTimestamp; // Agreement Created Time with Date (created, when
           contract is signed)
28
29     string public VehicleIdentificationNumber; // Unique identification of the vehicle
30     string public contractInPdf; // Give a unique link to ipfs site, where the AGB can
            be downloaded (open in Brave Browser)
31
32     uint public minimumBalanceRequired; //show minimum Balance required
33     // Time between payments is set in function showMinimumBalanceRequired()
34
35     uint public accumulatedPayment; // how much is already paid
36     bool public isContractSigned;   // has the lessee already signed the contract
37     bool public contractTerminated; // is the smart contract still active
38     string public comment; // This is a comment, which can be change, e.g. why the
           contract ended
39
40     // Set value: e.g. 1e18 in ETH, 1e15 is in finney, 1e9 in Gwei, 1e0 in Wei
41     uint constant value = 1e15;  // in the example we use finney
42
43
44     // This gives Boundary condition for the contract
45     constructor (address _recipient, string memory _currentOwner, string memory
           _VehicleIdentificationNumber, uint _leasingTotalCost, uint _singlePayment, uint
            _recurringPayment, string memory _contractInPdf)
46         {
47         // This Parametrs can be set during constructor (more flexibel duriong
               deployment)
48         recipient = _recipient;
49         currentOwner = _currentOwner;
50         leasingTotalCost = _leasingTotalCost * value ;     // for simplification we
               use finney (1e15)
51         singlePayment = _singlePayment * value;
52         recurringPayment = _recurringPayment * value;
53         VehicleIdentificationNumber = _VehicleIdentificationNumber;
54         contractInPdf = _contractInPdf;
55
56
57
58
59          /* // Here fix parameter in contract (its simpler to deploy like this the
                contract)
60         constructor () { //address _recipient, string memory _currentOwner, string
               memory _VehicleIdentificationNumber, uint _leasingTotalCost, uint
               _singlePayment, uint _recurringPayment, string memory _contractInPdf){
61         recipient = 0x000e4d3d97A8Edd4873763a9Fc83E2ff69DBfA30; //_recipient;
62         currentOwner = 'Kestenholz'; //_currentOwner;
63         leasingTotalCost = 10 * value; //_leasingTotalCost;    10 finney
64         singlePayment = 1 * value; //_singlePayment;
65         recurringPayment = 1 * value; //_recurringPayment;
66         VehicleIdentificationNumber = "WPO ZZZ 91 ZDS 102 886"; //
               _VehicleIdentificationNumber;
67         contractInPdf = "ipfs://..."; //_contractInPdf;
68         */
69
70
71
72
```

```
73
74          // Fixed Parameters
75          //timeBetweenPayment = 1 minutes; // This should be months in leasing contract
76          isContractSigned = false;
77          contractTerminated = false;
78      }
79
80
81
82      // sign contract with first payment
83      function signContract(string memory _newOwner) public payable{
84          require(!contractTerminated, "Contract is terminated");
85          require(!isContractSigned, "Contract is already signed");
86          require(msg.value>=singlePayment, "Insuficient Funds");
87          require(msg.value<=leasingTotalCost, "Payment surpasses leasing cost.");
88
89
90          accumulatedPayment += msg.value;    // add minimum of the singlePayment to
                accumulatedPayment
91          isContractSigned = true;            // Both side agree to the contract
92          newOwner = _newOwner;               // Give Name of new Owner
93          /*
94             Remark: - the person who signs the contract, does not have to pay.
95                     - multiple addresses can pay
96                     - no refund is possible
97          */
98
99          createdTimestamp = block.timestamp; // create tinestamp for recurringPayment
100     }
101
102
103     // send Payment
104     function sendPayment() public payable {
105         require(!contractTerminated, "Contract is terminated"); // can not do payment
                if contract is terminated
106         require(isContractSigned, "Contract is not signed");    // can not do payment
                if contract is not signed
107         require(accumulatedPayment+msg.value<=leasingTotalCost,"Accumulated payment
                would surpasses total leasing cost.");
108         accumulatedPayment += msg.value  ; // add payment to accumulatedPayment
109     }
110
111     // Check how much needs to be paid
112     function showMinimumBalanceRequired() public {
113         require(!contractTerminated, "Contract is terminated");
114         require(isContractSigned, "Contract is not signed");
115
116         uint numberOfTimeSteps; // how many recurennt Payment have already passed
117         /* Remark: - timestamp is in [s]. not optimal, since time of blocks can varry.
118                    - However, for months should be fine.        */
119         numberOfTimeSteps = (block.timestamp-createdTimestamp); // number of [s]
                passed
120         numberOfTimeSteps = numberOfTimeSteps / 60;             // convert here to
                minute
121         minimumBalanceRequired =  singlePayment+numberOfTimeSteps*recurringPayment; //
                 give lower boundry of the payment, which should have been done
122         if (minimumBalanceRequired>leasingTotalCost){   // ensure, that
                minimumBalanceRequired does not surpasses leasingTotalCost
123             minimumBalanceRequired = leasingTotalCost;
124         }
125
126     }
127
128
129     // if leasing Cost is paid, transfer ownership and withdraw money
130     function endContract() external {
131
132         require(!contractTerminated, "Contract is already terminated");
133
134         if (isContractSigned){
```

```
135             // if minimumBalanceRequired surpasses accumulatedPayment, it is possible
                    to end contract
136             showMinimumBalanceRequired();
137         }
138 
139         // Note: Everyone can currently end contract function
140         if (!isContractSigned) {
141             // when nobody signs the contract, the contract can be terminated
142             comment = "Contract was not signed and terminated";
143             contractTerminated = true;
144         } else if (leasingTotalCost==accumulatedPayment){
145             // Contract is signed and leasing cost is paid: transfer owner and
                    withdraw payment
146             oldOwner = currentOwner;
147             currentOwner = newOwner;
148             contractTerminated = true;
149             payable(recipient).transfer(accumulatedPayment); // payment of the
                    accumulatedPayment to seller
150             comment = "Contract succesfully executed";
151         } else if (minimumBalanceRequired>accumulatedPayment){
152             /*  - current Payment is not sufficent (lower the minimumBalanceRequired)
153                 - cancel contract, no owner transfer, withdraw payment to lessor    */
154             contractTerminated = true;
155             payable(recipient).transfer(accumulatedPayment);
156             comment = "Contract terminted: Leasing condition were not fullfiled.";
157         } else {
158             comment = "Condition to end contract are not met.";
159         }
160     }
161 }
```