Smart Contracts

Intoduction to Solidity

SENACOR

self_destruct(owner)

```
// this contract is used to make other contracts „killable"
contract mortal {
    /* Define variable owner of the type address*/
    address owner;

    /* this function sets the owner of the contract (at initi) */
    function mortal() { owner = msg.sender; }

    /* Function to recover the funds on the contract */
    function kill() { if (msg.sender == owner) selfdestruct(owner); }
}

contract greeter is mortal {
    /* see next slide ... */
```

Overview | Internal Transactions

The Contract Call **From** 0xf307ea98bedaa82… **To** 0x831d23cfb62c337… produced 1 Contract Internal Transaction :

Type_Index	From		To
✔ suicide_0	0x831d23cfb62c337…	→	0xf307ea98bedaa82…

SENACOR

Message (msg) Object, this, send()

```
// this contract is used to make other contracts „killable"
contract mortal {
    /* Define variable owner of the type address*/
    address owner;

    /* this function sets the owner of the contract (at initi) */
    function mortal() { owner = msg.sender; }

    /* Function to recover the funds on the contract */
    function kill() { if (msg.sender == owner) selfdestruct(owner); }
}
```

- msg.sender = the address of the account (Contract-Account or User-Account) that triggered the transaction.
- msg.value = the money amount set in the transaction.
- For retrieving the address of the current contract use the key-word **`this`**.
- To send money from one contract to another use the function **`send`**:

```
//(...)
msg.sender.send(amount);
//(...)
```

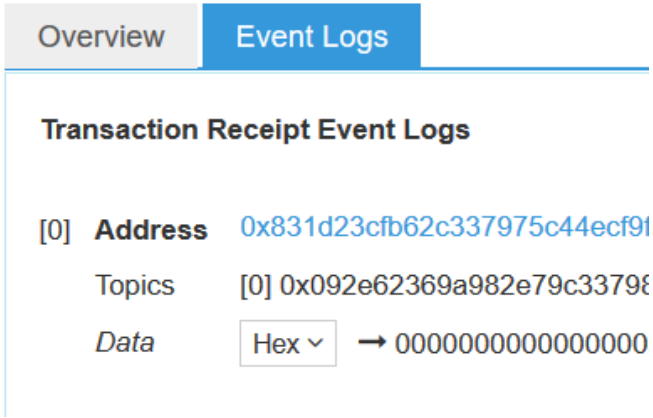
SENACOR

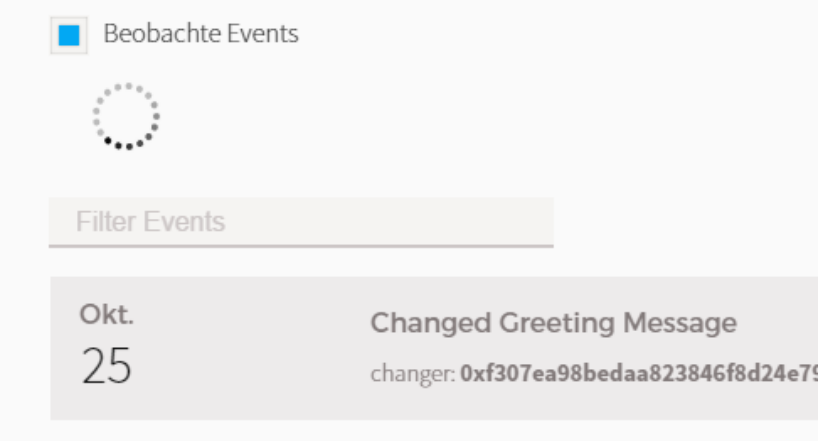
Events (Logging)

```
contract greeter is mortal {
    /* define variable greeting of the type string */
    string greeting;
    event ChangedGreetingMessage(address changer);

    // (...)

    /* change the greeting message */
    function changeGreeting(string _greeting) {
        greeting = _greeting;
        ChangedGreetingMessage(msg.sender);
    }
}
```





SENACOR

Conversion-Units (there are no floating point types)

```
/*  at initialization, setup the owner */
function Crowdsale(
    address ifSuccessfulSendTo,
    uint fundingGoalInEthers,
    uint durationInMinutes,
    uint etherCostOfEachToken,
    token addressOfTokenUsedAsReward
) {
    beneficiary = ifSuccessfulSendTo;
    fundingGoal = fundingGoalInEthers * 1 ether;
    deadline = now + durationInMinutes * 1 minutes;
    price = etherCostOfEachToken * 1 ether;
    tokenReward = token(addressOfTokenUsedAsReward);
}
```

Deadline

1477499824 *(in 4 Stunden)* = 4 hours in hundred milliseconds

Funding goal

20000000000000000000

SENACOR

Unnamed function (Fallback function)

```
function () payable {
    if (crowdsaleClosed) throw;

    uint amount = msg.value;
    balanceOf[msg.sender] += amount;
    amountRaised += amount;

    tokenReward.transfer(msg.sender, amount / price);
    FundTransfer(msg.sender, amount, true);
}
```

- Die empty function is the fallback function of the contract.
- This function is called when no other function matches the signature.
- The function can neither specify input parameters nor a return value.
- This function is used to accept money transactions to a contract (see „send()“).

SENACOR

Modifier (generell)

```
modifier afterDeadline() { if (now >= deadline) _; }
```

```
function checkGoalReached() afterDeadline {
    if (amountRaised >= fundingGoal){
        fundingGoalReached = true;
        GoalReached(beneficiary, amountRaised);
    }
    crowdsaleClosed = true;
}
```

- Modifiers are code pieces that are executed before a certain function is executed (similar to an aspect in AOP). Usually they contain check-criteria that define if a function is to be executed or not.
- When you decorate a function with a modifier the modifier will be called first upon function call. The modifier can then decide if the function‘s code should be executed or not.
- The execution of the function code is triggered by “_”.

SENACOR

Modifier: payable

```
function () payable {
    if (crowdsaleClosed) throw;

    uint amount = msg.value;
    balanceOf[msg.sender] += amount;
    amountRaised += amount;

    tokenReward.transfer(msg.sender, amount / price);
    FundTransfer(msg.sender, amount, true);
}
```

- payable is a special modifier that marks a function as money-transaction function. If you decorate a function with payable you can send ether to the contract through this function.
- Since solidity 0.4.x you also have to decorate the empty function with payable in order to let her accept money transactions to the contract.
- You can also decorate other functions than the empty function with payable. If you do this these functions will accept money transactions.

SENACOR

Exception Handling: require

```
bool public crowdsaleClosed;
// (...)

function () payable {
    require(!crowdsaleClosed);

    uint amount = msg.value;
    // (...)
}
```

- If require fails it will throw.

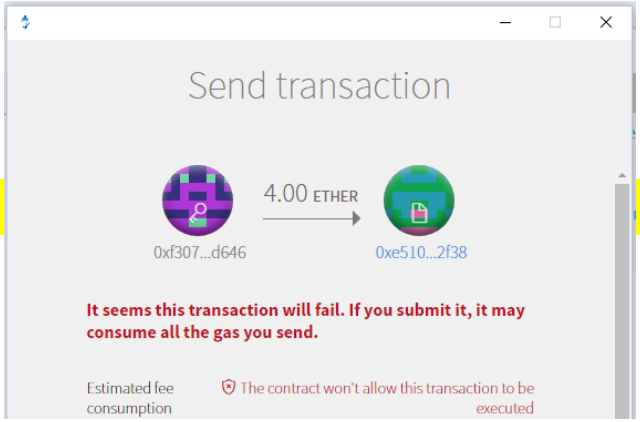
SENACOR

Exception Handling: throw (deprecated)

```
bool public crowdsaleClosed;
// (...)

function () payable {
    if (crowdsaleClosed) throw;

    uint amount = msg.value;
    // (...)
}
```



- Throw stops the execution of a contract (exception).
- Before you actually execute a function (when you prepare to send a transaction) the Mist client actually checks if the function will run into a „throw" with the settings you are calling it with. If a throw will be reached Mist will show a „standard" error:
  - *„It seems this transaction will fail. If you submit it, it my consum all the gas you send."*
    - *„The contract won't allow this transaction to be executed."*
- At the moment you cannot define any other exceptions. If you see this message you should not execute the transaction but you should find the error in your contract code.