# **7245**

# **CONTRACT ID: 0x6816dF892602c7E6b741f03FEf9c7eC57061A73f**

# Transaction 0x980caab45bd3478f9e7c307d398941d998f32636944cd687081d2747a0ca2c97

TxHash:

0x980caab45bd3478f9e7c307d398941d998f32636944cd687081d2747a0ca2c97

TxReceipt Status:

Success

Block Height:

[5083867](https://etherscan.io/block/5083867) (4 block confirmations)

TimeStamp:

1 min ago (Feb-13-2018 04:43:35 PM +UTC)

From:

[0x515581325a538a3dd7fccbe3a1cd6cf054f52a62](https://etherscan.io/address/0x515581325a538a3dd7fccbe3a1cd6cf054f52a62)

To:

[Contract [0x6816df892602c7e6b741f03fef9c7ec57061a73f](https://etherscan.io/address/0x6816df892602c7e6b741f03fef9c7ec57061a73f) Created]

Value:

0 Ether ($0.00)

Gas Limit:

1000000

Gas Used By Txn:

681565

Gas Price:

0**.**000000028 Ether (28 Gwei)

Actual Tx Cost/Fee:

0**.**01908382 Ether ($16.03)

Cumulative Gas Used:

7206853

Nonce:

14

Input Data:



**SMART CONTRACT**

contract Token {

/// @return total amount of tokens

function totalSupply() constant returns (uint256 supply) {}

/// @param \_owner The address from which the balance will be retrieved

/// @return The balance

function balanceOf(address \_owner) constant returns (uint256 balance) {}

/// @notice send `\_value` token to `\_to` from `msg.sender`

/// @param \_to The address of the recipient

/// @param \_value The amount of token to be transferred

/// @return Whether the transfer was successful or not

function transfer(address \_to, uint256 \_value) returns (bool success) {}

/// @notice send `\_value` token to `\_to` from `\_from` on the condition it is approved by `\_from`

/// @param \_from The address of the sender

/// @param \_to The address of the recipient

/// @param \_value The amount of token to be transferred

/// @return Whether the transfer was successful or not

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {}

/// @notice `msg.sender` approves `\_addr` to spend `\_value` tokens

/// @param \_spender The address of the account able to transfer the tokens

/// @param \_value The amount of wei to be approved for transfer

/// @return Whether the approval was successful or not

function approve(address \_spender, uint256 \_value) returns (bool success) {}

/// @param \_owner The address of the account owning tokens

/// @param \_spender The address of the account able to transfer the tokens

/// @return Amount of remaining tokens allowed to spent

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);

}

/\*

This implements ONLY the standard functions and NOTHING else.

For a token like you would want to deploy in something like Mist, see HumanStandardToken.sol.

If you deploy this, you won't have anything useful.

Implements ERC 20 Token standard: https://github.com/ethereum/EIPs/issues/20

.\*/

contract StandardToken is Token {

function transfer(address \_to, uint256 \_value) returns (bool success) {

//Default assumes totalSupply can't be over max (2^256 - 1).

//If your token leaves out totalSupply and can issue more tokens as time goes on, you need to check if it doesn't wrap.

//Replace the if with this one instead.

//if (balances[msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {

if (balances[msg.sender] >= \_value && \_value > 0) {

balances[msg.sender] -= \_value;

balances[\_to] += \_value;

Transfer(msg.sender, \_to, \_value);

return true;

} else { return false; }

}

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {

//same as above. Replace this line with the following if you want to protect against wrapping uints.

//if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {

if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && \_value > 0) {

balances[\_to] += \_value;

balances[\_from] -= \_value;

allowed[\_from][msg.sender] -= \_value;

Transfer(\_from, \_to, \_value);

return true;

} else { return false; }

}

function balanceOf(address \_owner) constant returns (uint256 balance) {

return balances[\_owner];

}

function approve(address \_spender, uint256 \_value) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

return true;

}

function allowance(address \_owner, address \_spender) constant returns (uint256 remaining) {

return allowed[\_owner][\_spender];

}

mapping (address => uint256) balances;

mapping (address => mapping (address => uint256)) allowed;

uint256 public totalSupply;

}

/\*

This Token Contract implements the standard token functionality (https://github.com/ethereum/EIPs/issues/20) as well as the following OPTIONAL extras intended for use by humans.

In other words. This is intended for deployment in something like a Token Factory or Mist wallet, and then used by humans.

Imagine coins, currencies, shares, voting weight, etc.

Machine-based, rapid creation of many tokens would not necessarily need these extra features or will be minted in other manners.

1) Initial Finite Supply (upon creation one specifies how much is minted).

2) In the absence of a token registry: Optional Decimal, Symbol & Name.

3) Optional approveAndCall() functionality to notify a contract if an approval() has occurred.

.\*/

contract HumanStandardToken is StandardToken {

function () {

//if ether is sent to this address, send it back.

throw;

}

/\* Public variables of the token \*/

/\*

NOTE:

The following variables are OPTIONAL vanities. One does not have to include them.

They allow one to customise the token contract & in no way influences the core functionality.

Some wallets/interfaces might not even bother to look at this information.

\*/

string public name; //fancy name: eg Simon Bucks

uint8 public decimals; //How many decimals to show. ie. There could 1000 base units with 3 decimals. Meaning 0.980 SBX = 980 base units. It's like comparing 1 wei to 1 ether.

string public symbol; //An identifier: eg SBX

string public version = 'H0.1'; //human 0.1 standard. Just an arbitrary versioning scheme.

function HumanStandardToken(

uint256 \_initialAmount,

string \_tokenName,

uint8 \_decimalUnits,

string \_tokenSymbol

) {

balances[msg.sender] = \_initialAmount; // Give the creator all initial tokens

totalSupply = \_initialAmount; // Update total supply

name = \_tokenName; // Set the name for display purposes

decimals = \_decimalUnits; // Amount of decimals for display purposes

symbol = \_tokenSymbol; // Set the symbol for display purposes

}

/\* Approves and then calls the receiving contract \*/

function approveAndCall(address \_spender, uint256 \_value, bytes \_extraData) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

//call the receiveApproval function on the contract you want to be notified. This crafts the function signature manually so one doesn't have to include a contract in here just for this.

//receiveApproval(address \_from, uint256 \_value, address \_tokenContract, bytes \_extraData)

//it is assumed that when does this that the call \*should\* succeed, otherwise one would use vanilla approve instead.

if(!\_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))), msg.sender, \_value, this, \_extraData)) { throw; }

return true;

}

}

CONTRACT ABI

[{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"type":"function"},{"constant":false,"inputs":[{"name":"\_spender","type":"address"},{"name":"\_value","type":"uint256"}],"name":"approve","outputs":[{"name":"success","type":"bool"}],"type":"function"},{"constant":true,"inputs":[],"name":"totalSupply","outputs":[{"name":"","type":"uint256"}],"type":"function"},{"constant":false,"inputs":[{"name":"\_from","type":"address"},{"name":"\_to","type":"address"},{"name":"\_value","type":"uint256"}],"name":"transferFrom","outputs":[{"name":"success","type":"bool"}],"type":"function"},{"constant":true,"inputs":[],"name":"decimals","outputs":[{"name":"","type":"uint8"}],"type":"function"},{"constant":true,"inputs":[],"name":"version","outputs":[{"name":"","type":"string"}],"type":"function"},{"constant":true,"inputs":[{"name":"\_owner","type":"address"}],"name":"balanceOf","outputs":[{"name":"balance","type":"uint256"}],"type":"function"},{"constant":true,"inputs":[],"name":"symbol","outputs":[{"name":"","type":"string"}],"type":"function"},{"constant":false,"inputs":[{"name":"\_to","type":"address"},{"name":"\_value","type":"uint256"}],"name":"transfer","outputs":[{"name":"success","type":"bool"}],"type":"function"},{"constant":false,"inputs":[{"name":"\_spender","type":"address"},{"name":"\_value","type":"uint256"},{"name":"\_extraData","type":"bytes"}],"name":"approveAndCall","outputs":[{"name":"success","type":"bool"}],"type":"function"},{"constant":true,"inputs":[{"name":"\_owner","type":"address"},{"name":"\_spender","type":"address"}],"name":"allowance","outputs":[{"name":"remaining","type":"uint256"}],"type":"function"},{"inputs":[{"name":"\_initialAmount","type":"uint256"},{"name":"\_tokenName","type":"string"},{"name":"\_decimalUnits","type":"uint8"},{"name":"\_tokenSymbol","type":"string"}],"type":"constructor"},{"anonymous":false,"inputs":[{"indexed":true,"name":"\_from","type":"address"},{"indexed":true,"name":"\_to","type":"address"},{"indexed":false,"name":"\_value","type":"uint256"}],"name":"Transfer","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"name":"\_owner","type":"address"},{"indexed":true,"name":"\_spender","type":"address"},{"indexed":false,"name":"\_value","type":"uint256"}],"name":"Approval","type":"event"}]

READ CONTRACT INFORMATION

1. name THOR string  
2. totalSupply 100000000000000000000000 uint256  
3. decimals 18 uint8  
4. version H0.1 string  
5. balanceOf   
 \_owner (address)  
 Query  
 balance uint256  
6. symbol THR string  
7. allowance   
 \_owner (address)  
,   
 \_spender (address)  
 Query  
 remaining uint256