

Smart Contracts - Standard Contracts (ERC-20)

Standard ERC-20 Contract

SimpleToken

Method Name	Const	\$	Params
Approval	event		(address owner, address spender, uint256 value)
INITIAL_SUPPLY	const		() returns (uint256)
Transfer	event		(address from, address to, uint256 value)
allowance	const		(address _owner, address _spender) returns (uint256)
approve	Tx		(address _spender, uint256 _value) returns (bool)
balanceOf	const		(address _owner) returns (uint256)
decimals	const		() returns (uint8)
decreaseApproval	Tx		(address spender, uint256 _subtractedValue) returns (bool)
increaseApproval	Tx		(address _spender, uint256 _addedValue) returns (bool)
name	const		() returns (string)
symbol	const		() returns (string)
totalSupply	const		() returns (uint256)
transfer	Tx		(address _to, uint256 _value) returns (bool)
transferFrom	Tx		(address from, address _to, uint256 _value) returns (bool)
constructor	()		

SimpleToken Ours derived from StandardToken

```
pragma solidity >=0.4.25 <0.7.0;

import "openzeppelin-solidity/contracts/token/ERC20/StandardToken.sol";

/**
```

```

* @title SimpleToken
* @dev Very simple ERC20 Token example, where all tokens are pre-assigned to the creat
* Note they can later distribute these tokens as they wish using `transfer` and other
* `StandardToken` functions.
*/
contract SimpleToken is StandardToken {

    string public constant name = "SimpleToken"; // solium-disable-line uppercase
    string public constant symbol = "SIM"; // solium-disable-line uppercase
    uint8 public constant decimals = 0; // solium-disable-line uppercase

    uint256 public constant INITIAL_SUPPLY = 10000 * (10 ** uint256(decimals));

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor() public {
        totalSupply_ = INITIAL_SUPPLY;
        balances[msg.sender] = INITIAL_SUPPLY;
        emit Transfer(0x0, msg.sender, INITIAL_SUPPLY);
    }
}

```

StandardToken

```

pragma solidity >=0.4.25 <0.7.0;

import "./BasicToken.sol";
import "./ERC20.sol";

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
 * Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master/smart
 */
contract StandardToken is ERC20, BasicToken {

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

    /**
     * @dev Transfer tokens from one address to another
     * @param _from address The address which you want to send tokens from
     * @param _to address The address which you want to transfer to
     */

```

```
* @param _value uint256 the amount of tokens to be transferred
*/
function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
    public
    returns (bool)
{
    require(_to != address(0));
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);
    return true;
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf
 * Beware that changing an allowance with this method brings the risk that someone ma
 * and the new allowance by unfortunate transaction ordering. One possible solution t
 * race condition is to first reduce the spender's allowance to 0 and set the desired
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param _spender The address which will spend the funds.
 * @param _value The amount of tokens to be spent.
 */
function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param _owner address The address which owns the funds.
 * @param _spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
function allowance(
    address _owner,
    address _spender
)
    public
    view
    returns (uint256)
{
    return allowed[_owner][_spender];
}
```

```
}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
function increaseApproval(
    address _spender,
    uint256 _addedValue
)
    public
    returns (bool)
{
    allowed[msg.sender][_spender] = (
        allowed[msg.sender][_spender].add(_addedValue));
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseApproval(
    address _spender,
    uint256 _subtractedValue
)
    public
    returns (bool)
{
    uint256 oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
        allowed[msg.sender][_spender] = 0;
    } else {
        allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}
```

```
}
```

BasicToken

```
pragma solidity >=0.4.25 <0.7.0;

import "./ERC20Basic.sol";
import "../math/SafeMath.sol";

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
    using SafeMath for uint256;

    mapping(address => uint256) balances;

    uint256 totalSupply_;

    /**
     * @dev Total number of tokens in existence
     */
    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    /**
     * @dev Transfer token for a specified address
     * @param _to The address to transfer to.
     * @param _value The amount to be transferred.
     */
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    /**
     * @dev Gets the balance of the specified address.

```

```
* @param _owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/
function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
}

}
```

ERC20

```
pragma solidity >=0.4.25 <0.7.0;

import "./ERC20Basic.sol";

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender)
        public view returns (uint256);

    function transferFrom(address from, address to, uint256 value)
        public returns (bool);

    function approve(address spender, uint256 value) public returns (bool);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
}
```