



OST

Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

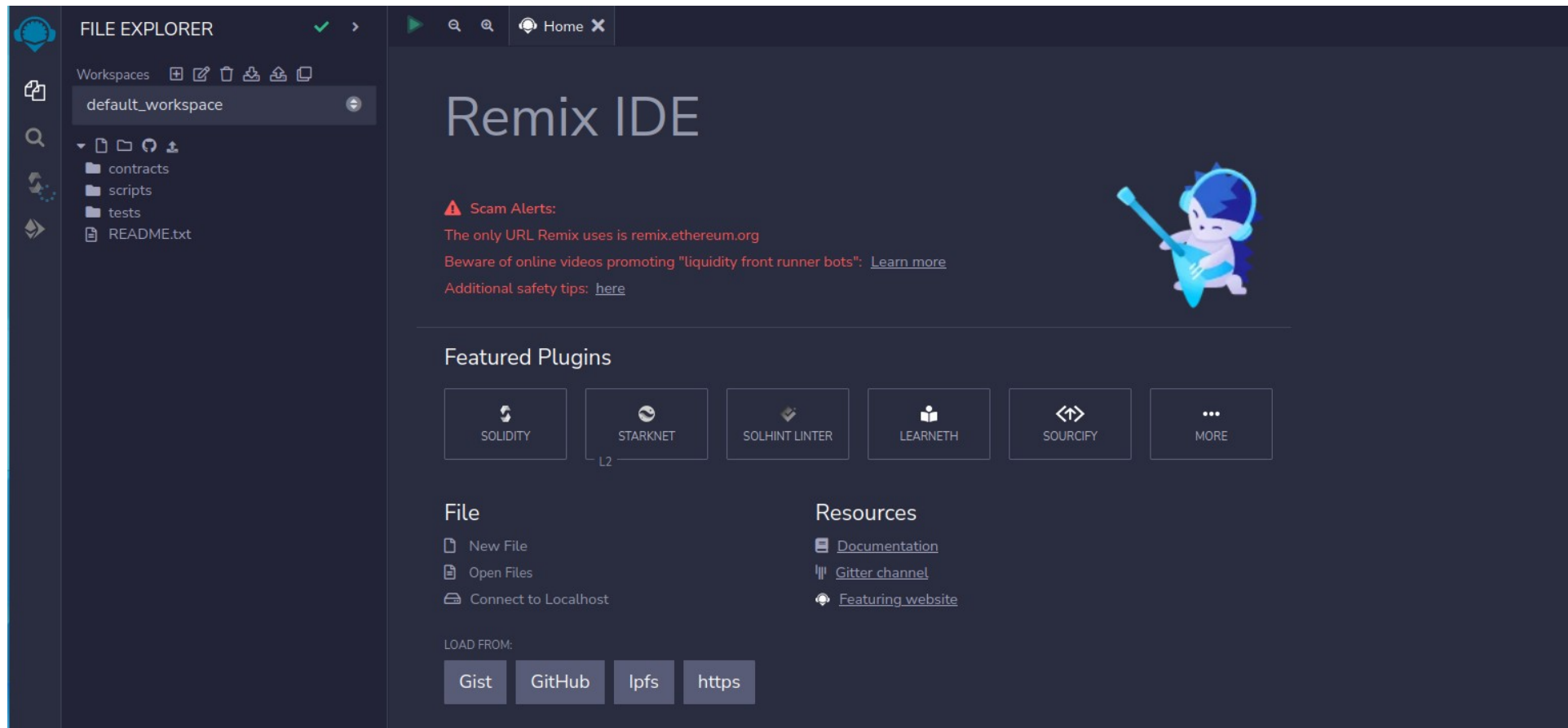
Solidity

Thomas Bocek

09.10.2022

Solidity IDE

- <https://remix.ethereum.org>
- In combination with Solidity IntelliJ plugin (not ideal)



Solidity - <http://solidity.readthedocs.io>

- Version Pragma

```
pragma solidity ^0.4.24; // not before 0.4.24, not after 0.5.0
```

- Comments

```
// This is a single-line comment.
```

```
/*  
This is a  
multi-line comment.  
*/
```

- Data Types

byte, bytes2 to bytes32, bytes (same as byte[], but more expensive), string, int8 to uint256, address, enum, bool

- Simple contract

```
contract SimpleStorage {  
    uint256 storedData; // State variable  
}
```

<https://consensys.github.io/smart-contract-best-practices/>

<https://learnxinyminutes.com/docs/solidity/>

<https://consensys.net/blog/developers/solidity-best-practices-for-smart-contract-security/>



Solidity IDE

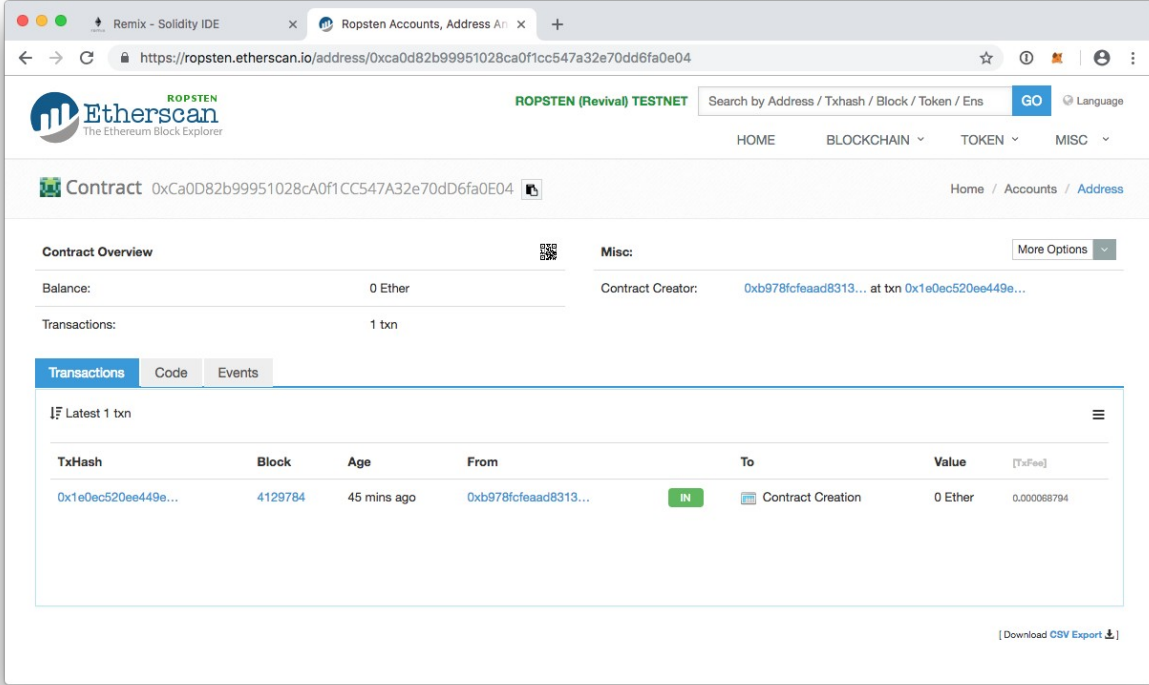
- Create your first contract

```
pragma solidity ^0.8.0;  
// Minimal contract example  
contract SimpleStorage {  
    uint256 storedData; // State variable  
}
```

- Install MetaMask
- Compile
- Compile and push «Deploy»

Check Deployment

- You have mined your first contract!



The screenshot shows the Etherscan Ropsten Testnet interface for a contract address: `0xCa0d82b99951028ca0f1cc547a32e70dd6fa0e04`. The page displays the contract overview, including the balance (0 Ether) and a single transaction. The transaction details are as follows:

TxHash	Block	Age	From	To	Value	[TxFee]
0x1e0ec520ee449e...	4129784	45 mins ago	0xb978fcfeaad8313...	IN Contract Creation	0 Ether	0.000068794

Additional details visible on the page include the contract creator's address `0xb978fcfeaad8313... at txn 0x1e0ec520ee449e...` and a [Download CSV Export] link at the bottom right.

Gas: Ethereum's Fuel

- Price that is paid for running a transaction or a contract
- Unit of measuring computational work
- Every instruction needs to be paid for
- If you run out of gas, state is reverted, ETH gone

APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero} .
G_{base}	2	Amount of gas to pay for operations of the set W_{base} .
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low} .
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid} .
G_{high}	10	Amount of gas to pay for operations of the set W_{high} .
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the <i>Homestead transition</i> .
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

$W_{zero} = \{\text{STOP, RETURN}\}$

$W_{base} = \{\text{ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, POP, PC, MSIZE, GAS}\}$

$W_{verylow} = \{\text{ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORES, PUSH*, DUP*, SWAP*}\}$

$W_{low} = \{\text{MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}\}$

$W_{mid} = \{\text{ADDMOD, MULMOD, JUMP}\}$

$W_{high} = \{\text{JUMPI}\}$

$W_{extcode} = \{\text{EXTCODESIZE}\}$

Solidity - <http://solidity.readthedocs.io>

- Structs

```
struct Account {  
    string addr;  
    uint256 amount;  
}
```

Key	Value
0x3f51...	100
0x17aa...	0
0x4eb2...	85
...	...

- Mapping (key-value pair, not iterable, but **can be implemented**), think of Map<K,V> in Java or Dictionary<K,V> in C#
- mapping(address => uint256) public balances;
- balances[msg.sender] = 100; // Set balance of sender
- uint256 balance = balances[msg.sender]; // Get balance of sender
- delete(balances[msg.sender]);

Solidity – Functions

- «Standard» Functions: Can read and modify the state

```
uint256 counter;  
function setCounter(uint256 _newValue) public {  
    counter = _newValue;  
}
```

- View Functions: Do not modify the state (it's free!)

```
uint256 counter;  
function getCounter() public view returns (uint256) {  
    return counter;  
}
```

- Pure Functions: Do not read from or modify the state.

```
function multiply(uint256 a, uint256 b) public pure returns (uint256) {  
    return a * b;  
}
```


Solidity – Functions

- Internal Functions: Only callable internally

```
uint256 counter;  
function setCounter(uint256 _newValue) internal {  
    counter = _newValue;  
}
```

- External Functions: Only callable externally
Note: public and external differs in terms of gas usage

```
function setValues(uint256[20] _values) external {  
    // do something  
}
```

- Payable Functions: Receives plain Ether

```
function deposit() payable {  
    // Access msg.value to get amount of Ether  
}
```

Solidity – Basics

- Often used **Special Variables and Global Functions**
 - `block.number` (type of `uint256`): current block number
 - `gasleft()` (returns `uint256`): remaining gas
 - `msg.sender` (type of `address`): sender of the message
 - `msg.value` (type of `uint256`): number of wei sent with the message
 - `block.timestamp` (type of `uint256`): current block timestamp

Solidity – Basics

- Use require to test user inputs

```
if (msg.sender != owner) { throw; }
```

```
// Do something only the owner is allowed to
```

- behaves the same as:

```
require(msg.sender == owner);
```

```
// Do something only the owner is allowed to
```

```
//better: require(msg.sender == owner, "Unauthorized");
```

- OpenZeppelin: **Ownable**
- **The use of revert(), assert(), and require() in Solidity**
 - Require, when errors can happen will not use all gas
 - Assert, when errors should not happen, will use all gas

Solidity – Events/Notifications

- Events are a way for smart contracts written in Solidity to log that something has occurred
- Interested observers, notably JavaScript front ends for decentralized apps, can watch for events and react accordingly.

transaction cost	43044 gas
execution cost	21580 gas
hash	0x306ba94b3681cc650cf62d55ae4a121aea240a5dd7077cb660c03f77a82ae537
input	0x82a...00064
decoded input	<pre>{ "uint256 newBalance": "100" }</pre>
decoded output	<pre>{}</pre>
logs	<pre>[{ "from": "0x692a70d2e424a56d2c6c27aa97d1a86395877b3a", "topic": "0x5f66d2a93b609bc6596b75c6dbb0e4f3f7cafd4b3b617157ff304d1076e58375", "event": "Update", "args": { "0": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "1": "100", "_user": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "_newBalance": "100", "length": 2 } }]</pre>
value	0 wei

Solidity – Events/Notifications

```
contract EventsExample {  
    event OwnerChanged(address _oldOwner, address _newOwner);  
  
    function transfer(address _newOwner) public {  
        require(owner == msg.sender, "Sender not authorized");  
        emit OwnerChanged(owner, _newOwner);  
        owner = _newOwner;  
    }  
}
```

Numbers, Loops, Other contracts

- Before 0.7, SafeMath was essential, as overflow was not checked.
 - Many mistakes were made due to overflow/underflow in smart contracts
- After Solidity 0.7, overflow throws an error
 - `uint256(10) – uint256(11) → underflow, throws`
- if/else if/else
 - As with any other language
- Basic logic if/else, for, while, break, continue, return - no switch
- External contracts
 - Define interface e.g., `ExternalContract` with function `test()`
 - `ExternalContract(addr).test();`
- Transfer funds
 - `payable(this).transfer(1 ether);`
 - `payable(this).call.value(1 ether);`
 - Check return value
 - Gas is adjustable

```
for(uint256 x = 0; x < refundAddressList.length; x++) {  
    refundAddressList[x].transfer(SOME_AMOUNT);  
}
```

Modifier / Inheritance

- Modifiers can be used in functions

- Modifiers vs. require

```
modifier test(address adr) {  
    require(adr == _stored);  
    _;  
}
```

- `_` will be replaced with actual code of the function (...)
- function `contribute()` public payable `test(0x...) returns(uint256 id) {...}`

- Opinion: use well known modifiers, but rest do with require. Otherwise you need to search conditions some place else

- Inheritance, e.g., use OpenZeppelin contracts as base

```
contract Owned {  
    constructor() { owner = payable(msg.sender); }  
    address payable owner;  
}  
  
contract Test is Owned {  
    //virtual means, can be override  
    function setAnyOwner() virtual public {  
        owner = msg.sender;  
    }  
}
```