

Tutorial 1

Ethereum Remix IDE and Test Network Tutorial

This document contains step-by-step instructions for editing, testing, deploying, and interacting with smart contracts on the Ethereum test network.

Requirements:

- Google Chrome or Firefox

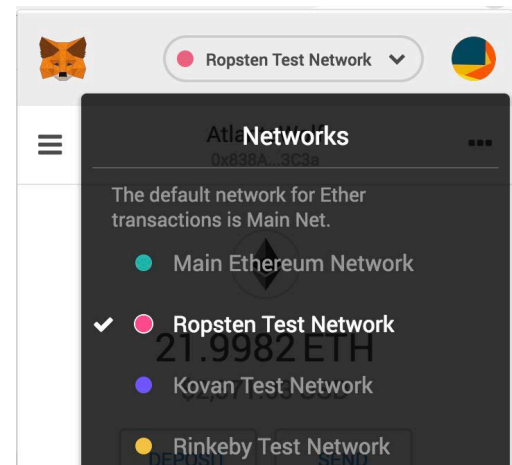
Step 1. Install MetaMask and create a new account

<https://metamask.io/>

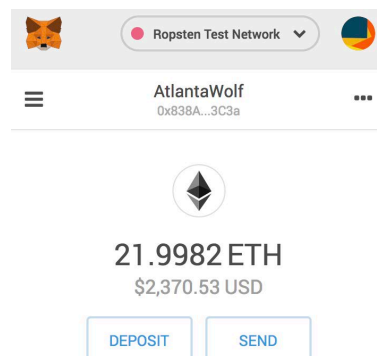
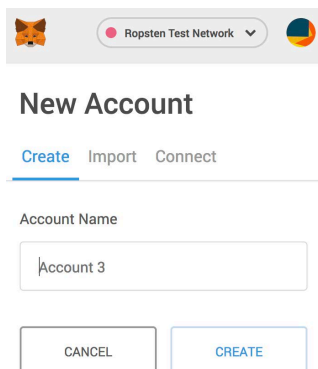
MetaMask is an Ethereum light-weight wallet implemented in a google chrome and FireFox plugin

After installing the MetaMask plugin, you will need to create a password. (Just for testing in this exercise, I used an insecure password and did not bother to back up the private key--make sure you'll do it) You can launch the MetaMask plugin by clicking the little orange fox icon that should appear in your extensions bar.

Make sure to switch to Ropsten Test network (click the little red circle in the top left corner, by default it will say Main Ethereum Network)



Step 2. Create an account/address, and collect testnet coins.



You can get test ether from your Metamask: Deposit> under Test Faucet click on "GET ETHER"

Or Copy the public key to the clipboard, and paste it in *Ropsten test coin yourself by using the testnet faucet.* <http://faucet.ropsten.be:3001/>

After receiving a transaction, you should see the balance indicator in MetaMask increase. You can also send transactions to other users/accounts directly from MetaMask by pressing the Send button. Once a transaction is sent, you'll be able to view it on Etherscan.

Step 3. Load a “Hello World” contract in the Remix IDE

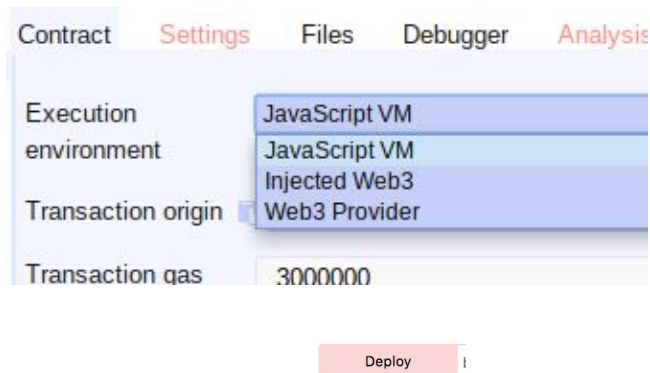
This tutorial will make use of Remix, and in-browser Solidity editor that lets you edit and test contract code. Let's load a sample file contract into the Remix IDE. Clicking the following link will load up the IDE and import some code for a simple test contract I created (it may appear under gist): <http://remix.ethereum.org/#gist=d9a5eefb68b83c48b196b5be65f1be54> (after loading to your Remix, just change the version of compiler to 0.4.11 manually--like below snapshot)

This is a crude voting style contract. Read it to get a sense of what it does. Whoever creates the contract is its “owner”. The owner can at any time add a “proposal”, which comes with a string description. Anyone can add votes to a proposal (in fact you can vote for any proposal as many times as you want). That's it!

In the IDE, the buttons on the left pane will also let you create new files (+ icon), or upload them from your computer (folder icon). The “Settings” tab on the right pane will also let you save your contracts to a gist on github.

```
+ gist/testcontract.sol x
1 pragma solidity ^0.4.11;
2 contract TestContract {
3
4     struct Proposal {
5         uint voteCount;
6         string description;
7     }
8
9     address public owner;
10    Proposal[] public proposals;
```

You can edit the program code for the contract in the browser. As you edit, the Solidity compiler (running in your browser) will automatically compile the code and show you any warning / errors it finds.



The Remix IDE provides a few ways of testing the contract. One way is by creating an instance of the contract directly in your browser. We'll try this first. **Click on the “Execution environment” and make sure “JavaScript VM” is selected-- this mode is more of local testing and deploying (give us a simulated blockchain inside our browser).**

Click the “Deploy” button to create an instance of the contract in the Javascript VM. You'll see some information about your newly created contract in the interaction/web interface pane under **“Deployed Contracts”**

The blue buttons let you query the current values of the public read-only fields of this contract. The “owner” corresponds to your address (not the testnet address you created earlier, just some temporary address made in the test environment for now--under **Account**--you can change it though) The array of proposals is empty, so we’ll get to that later.

Account  0xca3...a733c (99.9999999999984729   

▼ browser/testcontract.sol:TestContract at 0x692...77b3a (memory)  x

proposals uint256

owner 0: address: 0xca35b7d915458ef540ade6068dfe2f44e8fa733c

The pink buttons below let you interact with the test contract. Let’s create a new proposal. Type something in for the string parameter, and then click the “createProposal” button. **Don’t forget to include quotes “ ” for a string parameter.**

createProposal "This is a test proposal"

Now that we’ve created a proposal, we can query the “proposals” field by index. Make sure to enter an index into the field next to the blue “proposals” button or else you’ll see an error. If you’ve been following along, index 0 should show you the following returned fields. The description should match what you typed in, and the voteCount should be 0.

proposals 0

0: uint256: voteCount 0

1: string: description This is a test proposal

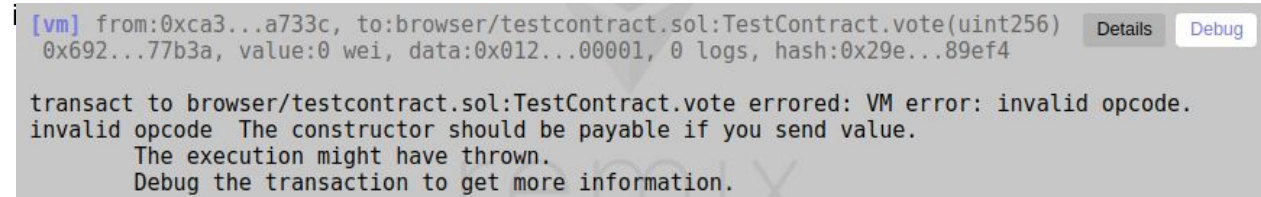
Outputs queries / transactions are also shown in the console:

```
call to browser/testcontract.sol:TestContract.proposals
[call] from: - , to: browser/testcontract.sol:TestContract.proposals(uint256), data: 013cf...00000, return:
{
  "0": "uint256: voteCount 0",
  "1": "string: description This is a test proposal"
}
```

Exercises: Tinker around, and try to figure out how to accomplish the next few steps yourself.

- Try to cast a vote for your proposal. (To vote for a proposal you need to pass in the index of the proposal.)
- Try to create additional proposals and two more voters. If you try to cast a vote for a proposal that doesn't exist yet, you'll see an error message in the console (though not a particularly



```
[vm] from:0xca3...a733c, to:browser/testcontract.sol:TestContract.vote(uint256)
0x692...77b3a, value:0 wei, data:0x012...00001, 0 logs, hash:0x29e...89ef4

transact to browser/testcontract.sol:TestContract.vote errored: VM error: invalid opcode.
invalid opcode The constructor should be payable if you send value.
The execution might have thrown.
Debug the transaction to get more information.
```

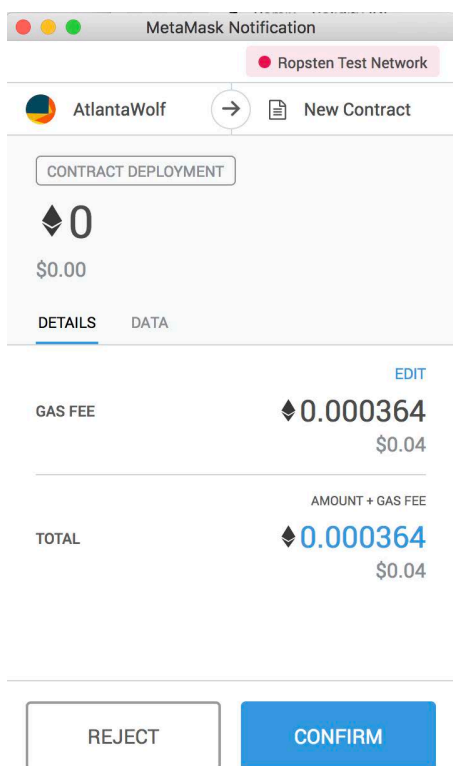
- Edit the contract code a little bit so it has some different behavior. For example, restrict the number of votes to some “maximum” value. Or keep track of who has already voted, so that each address can only vote once. Create an instance of a contract with the new code and test it.
- Click on the “Debug” button and poke around with the debugger.

Poke around with the Remix IDE for a bit before going on. The workflow we've just used is the is the easiest way to try out your contract, make sure it compiles, and make sure it behaves basically the way you expect.

Step 4. Publish the hello world contract to the Testnet (deploy the contract)

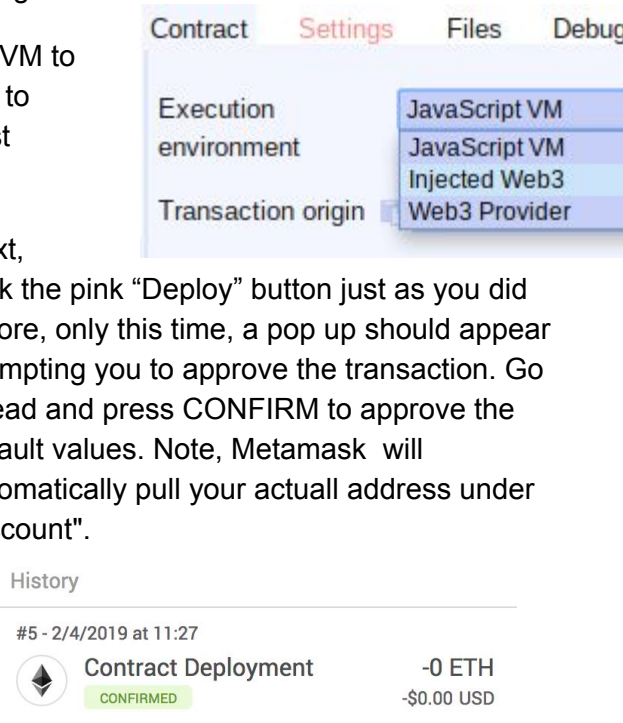
Let's take off the training wheels (well, sort of!) and get on the road.

Switch the Execution environment from JavaScriptVM to "Injected Web3". This will connect your Remix IDE to MetaMask, and therefore to the Ropsten public test network.



Next,

click the pink "Deploy" button just as you did before, only this time, a pop up should appear prompting you to approve the transaction. Go ahead and press CONFIRM to approve the default values. Note, Metamask will automatically pull your actual address under "Account".



After the popup disappears, you should be able to see your transaction show up in the "HISTORY" panel in the MetaMask plugin. Clicking on the transaction will take you to the Etherscan page for your transaction.

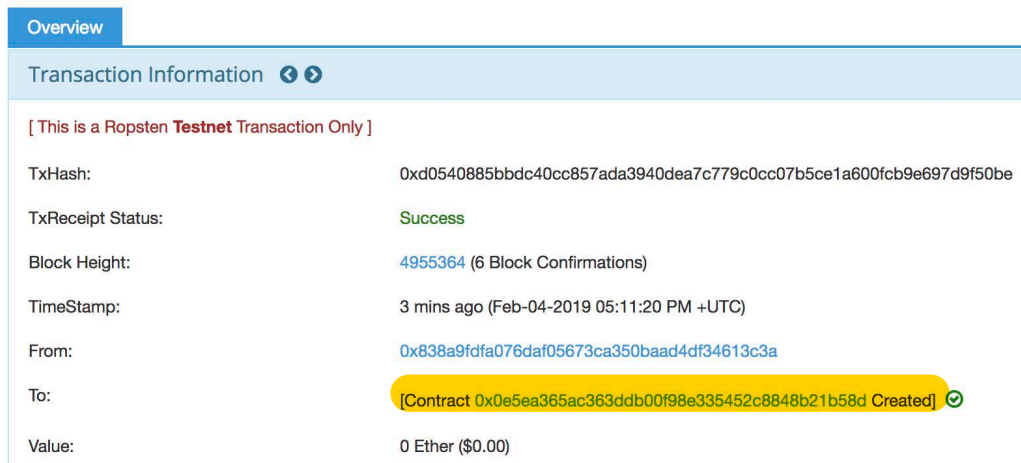
Go back to Remix, next you can follow the same instructions as before in Step 2 in order to interact with your contract, to invoke the "createProposal" method, the "vote" method, and view the public fields. Each time you try to invoke a method, it will pass it through to MetaMask, which will pop up a window asking you to. You will notice your MetaMask balance slowly decrease as you make transactions, since you are spending gas, and paying for it using testnet coins.

Deployed Contracts



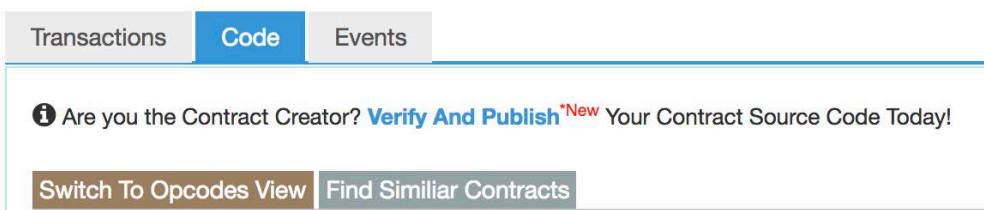
Step 5. Add the source code for your contract on Etherscan.

So, you've created a contract on the public test network. Great! But now want others to be able to use it too. Start by navigating to the Etherscan page for the address of the contract you created. You can get to this by clicking on the entry in your MetaMask "HISTORY" window, and then clicking on the "[Contract 0xXXXXXXX Created]" entry as highlighted below.



The screenshot shows the Etherscan 'Transaction Information' page for a contract deployment. At the top, it says '#5 - 2/4/2019 at 11:27' and 'Contract Deployment' with a 'CONFIRMED' status. The transaction value is '-0 ETH' and '-\$0.00 USD'. The 'Details' section shows 'From: 0x838A9FDA076D...' and 'New Contract'. The 'Overview' tab is selected, showing transaction details: TxHash (0xd0540885bbdc40cc857ada3940dea7c779c0cc07b5ce1a600fcb9e697d9f50be), TxReceipt Status (Success), Block Height (4955364, 6 Block Confirmations), TimeStamp (3 mins ago), From (0x838a9dfa076daf05673ca350baad4df34613c3a), To ([Contract 0x0e5ea365ac363ddb00f98e335452c8848b21b58d Created]), and Value (0 Ether (\$0.00)). A red warning message states '[This is a Ropsten Testnet Transaction Only]'.

Next click the "Contract Code" tab, and then click the "Verify And Publish" link.



The screenshot shows the 'Contract Code' tab selected in the Etherscan interface. It features a notification: 'Are you the Contract Creator? Verify And Publish*New Your Contract Source Code Today!'. Below this are two buttons: 'Switch To Opcodes View' and 'Find Similiar Contracts'.

What you should do next is copy and paste the solidity code from the Remix IDE into the Etherscan page where it says "Enter the Solidity Contract Code below". You will also need to give the contract a name. **Make sure to set the "Compiler Version" to match the version in your Remix-IDE.** You can find this at the end of the url in your Remix IDE tab. At the time of writing, the default is "`v0.4.11+commit.68ef5810.js`". Also make sure to set Optimization to disabled (unless you changed this setting in Remix, the point is it must match). Etherscan will then attempt to compile your Solidity code, and if it matches exactly the bytecode in the testnet blockchain, you'll get a thumbs up.

👍 Successfully generated ByteCode and ABI for Contract Address [\[0x0e5ea365ac363ddb00f98e335452c8848b21b58d\]](#)

Now when you navigate back to the Etherscan page for your contract, you'll see that the "Contract Source" tab and the "Read Contract" tab is available. The Read Contract tab gives a similar view as the panel in the Remix IDE, so you can see e.g. the proposals and the owner address. If you modified your contract to create other public fields, then they'd show up here too.

Transactions

Code 

Read Contract

Write Contract Beta

Events

 Read Contract Information

1. proposals

<input> (uint256)

<input> (uint256)

Query

voteCount *uint256*, description *string*

2. owner

0x838a9fdfa076daf05673ca350baad4df34613c3a *address*

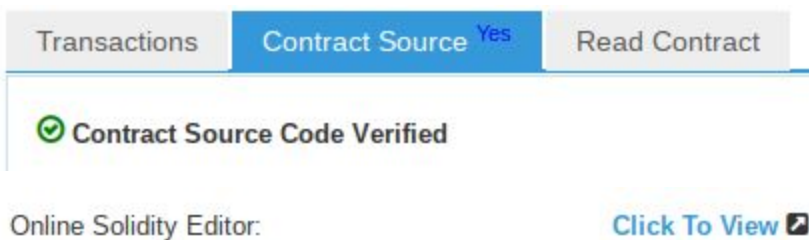
On Etherscan (ropsten), you can search for your verified deployed contract by name: <https://ropsten.etherscan.io/contractsVerified>

Step 6. Share your contract with someone else, or use someone else's contract.

Once you publish a contract on the Ethereum test network, and load up Etherscan with the verified source code, you can easily share your contract with them just by showing them the Etherscan URL or the contract address on its own.

Sharing via Verified source code:

To import into Remix a contract you received from someone else via an Etherscan URL, simply click the "Contract Source" tab, and then the "Click To View" link.



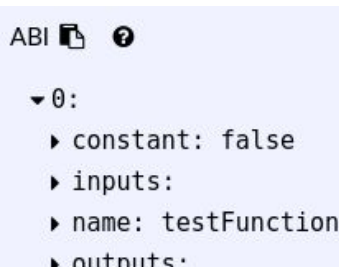
This will import the contract code into the Remix IDE (if "click to view" is not shown, simply copy the code and paste it in your Remix). Finally we have to do an extra step: click

the  button in the Remix IDE, and a popup will appear where you need to paste the address of the contract, starting with 0x. You can copy this easily from the Etherscan URL.

Then, from here you can query the contract or send it transactions.

Sharing via ABI:

To interact with someone else's contract without compiling the entire source code, you can



share the ABI (Application Binary Interface). You can get the ABI by clicking the "Details" button in the "Compile" tab, the clipboard icon copies it.

Once you have an ABI, you can paste it into MyEtherWallet to interact with a contract.

Contract Address

```
0x1c8f18ba95e74cc99f5db5ea36d945a834f29375
```

ABI / JSON Interface

```
[
  {
    "constant": false,
    "inputs": [ ]
```

Glossary / References

Overview of the services we'll use:

- Ropsten test network.

The public test network of Ethereum. (There are alternatives, this is the currently best-supported one).

- MetaMask: <https://metamask.io/>

a wallet for Ethereum you can install as a Firefox plugin or a Google Chrome plugin. This will hold your private key, and you can use it to approve transactions. It connects to a third-party service (run by <https://infura.io/>) as a bridge to Rinkeby testnetwork peer to peer network.

- Etherscan Block explorer: <https://ropsten.etherscan.io/>

Shows you the status of transactions and contracts on the Rinkeby blockchain.

You can use its "Verified and Publish" feature to register the source code for a contract you publish. This allows it to show you to read the data stored in the contract.

- Remix IDE <https://remix.ethereum.org/>

An in-browser Solidity editor that lets you edit and test contract code. It can also interoperate with MetaMask to interact with contracts on the Ethereum network. It can generate transactions, which you can then approve and send through MetaMask.

- Ropsten faucet

A website that gives you free coins on the test network