Remix Documentation

Release 1

yann300

Oct 20, 2020

New Layout Intro

Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Written in JavaScript, Remix supports both usage in the browser and locally.

Remix also supports testing, debugging and deploying of smart contracts and much more.

Our Remix project with all its features is available at remix.ethereum.org and more information can be found in these docs. Our IDE tool is available at our GitHub repository.

This set of documents covers instructions on how to use Remix and some tutorials to help you get started.

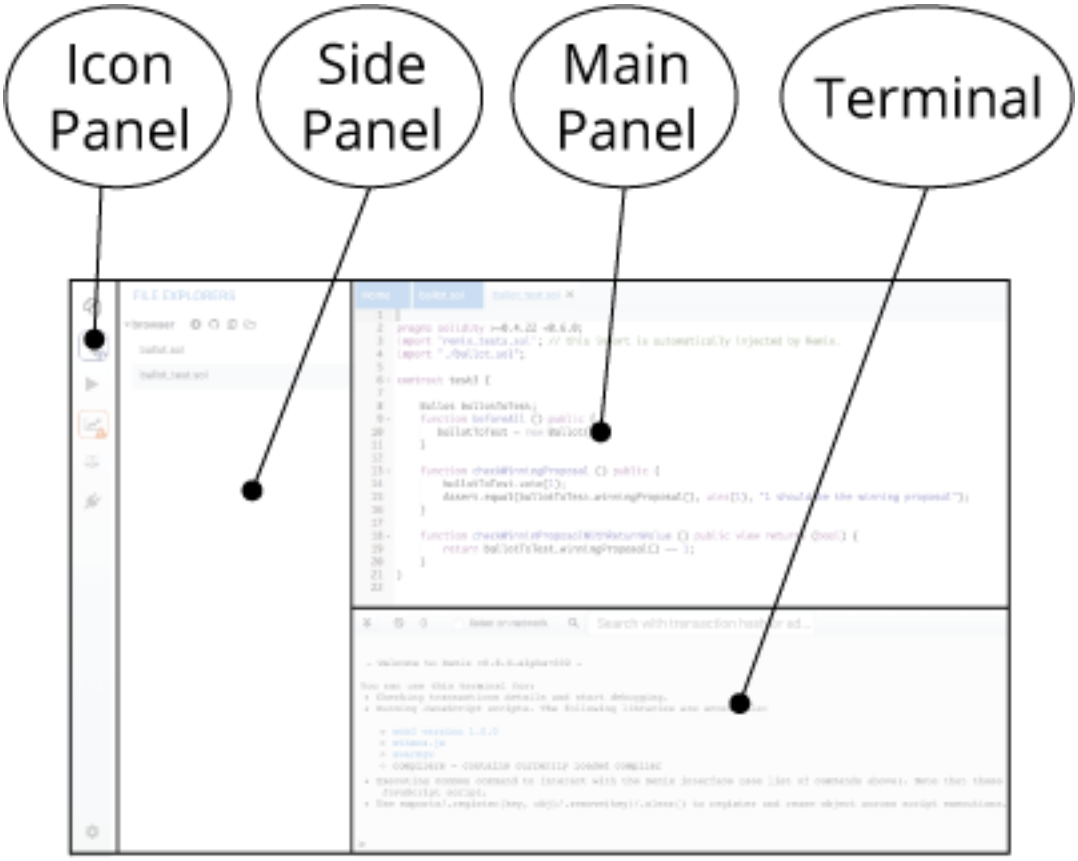
Useful links:

- Solidity documentation
- Remix alpha - The version where we test new Remix release (not stable!).
- Remix on Medium
- Ethereum StackExchange for Remix
- Community support channel
- Đapp Developer resources (Ethereum wiki)

CHAPTER 1

Remix-IDE Layout

1.1 The new structure

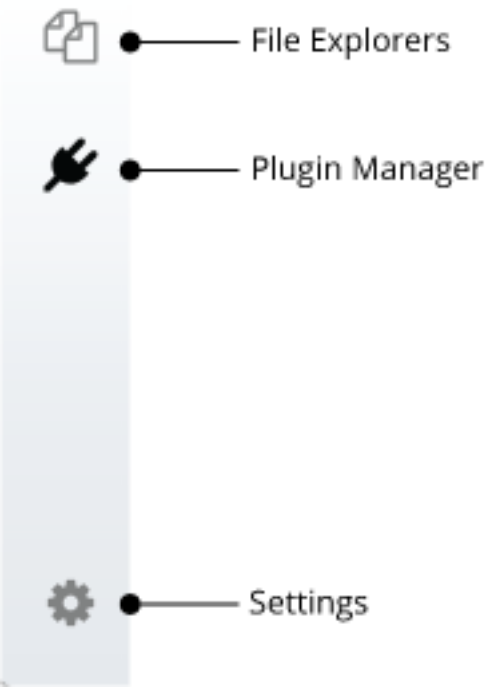


1. Icon Panel - click to change which plugin appears in the Side Panel

2. Side Panel - Most but not all plugins will have their GUI here.
3. Main Panel - In the old layout this was just for editing files. In the tabs can be plugins or files for the IDE to compile.
4. Terminal - where you will see the results of your interactions with the GUI's. Also you can run scripts here.

1.2 Icon Panel at Page Load

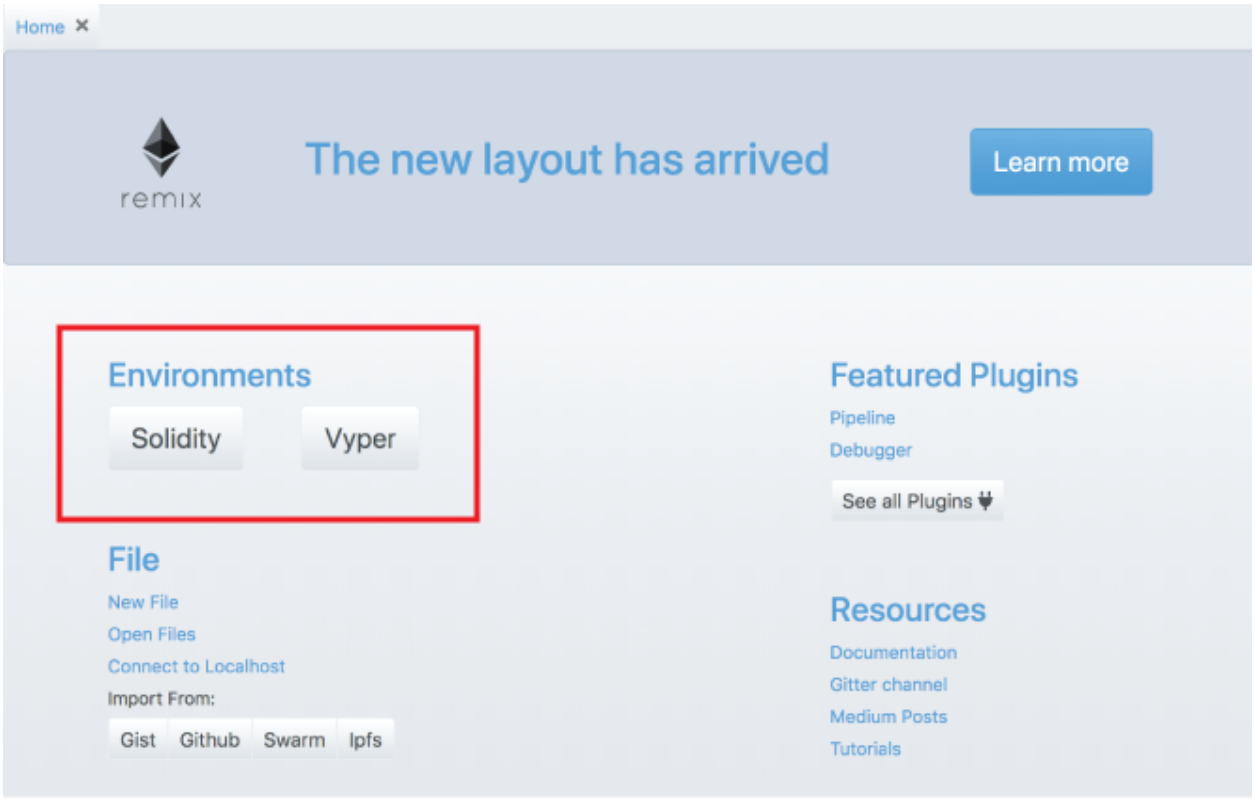
When you load remix - the icon panel show these icons by default.



Everything in remix is now a plugin... so the ***Plugin Manager*** is very important. In the old layout, each basic task in remix was separated into the tabs. Now these tabs are plugins.

But to activate a half a dozen plugins - (or however many you are using) each time the page load is **tedious**. So learn about the ***Environments***.

1.3 Homepage



The homepage is located in a tab in the Main Panel.

You can also get there by clicking the remix logo at the top of the icon panel.

1.3.1 Environments

Clicking on one of the environment buttons loads up a collection of plugins. We currently have a **Solidity** Button and a **Vyper** button. In the future you will be able to save your own environment.

To see all the plugins go to the **plugin manager** - by selecting the plug in the icon panel. 

The environment buttons are time & sanity savers - so you don't need to go to the plugin manager to get started everytime you load the page.

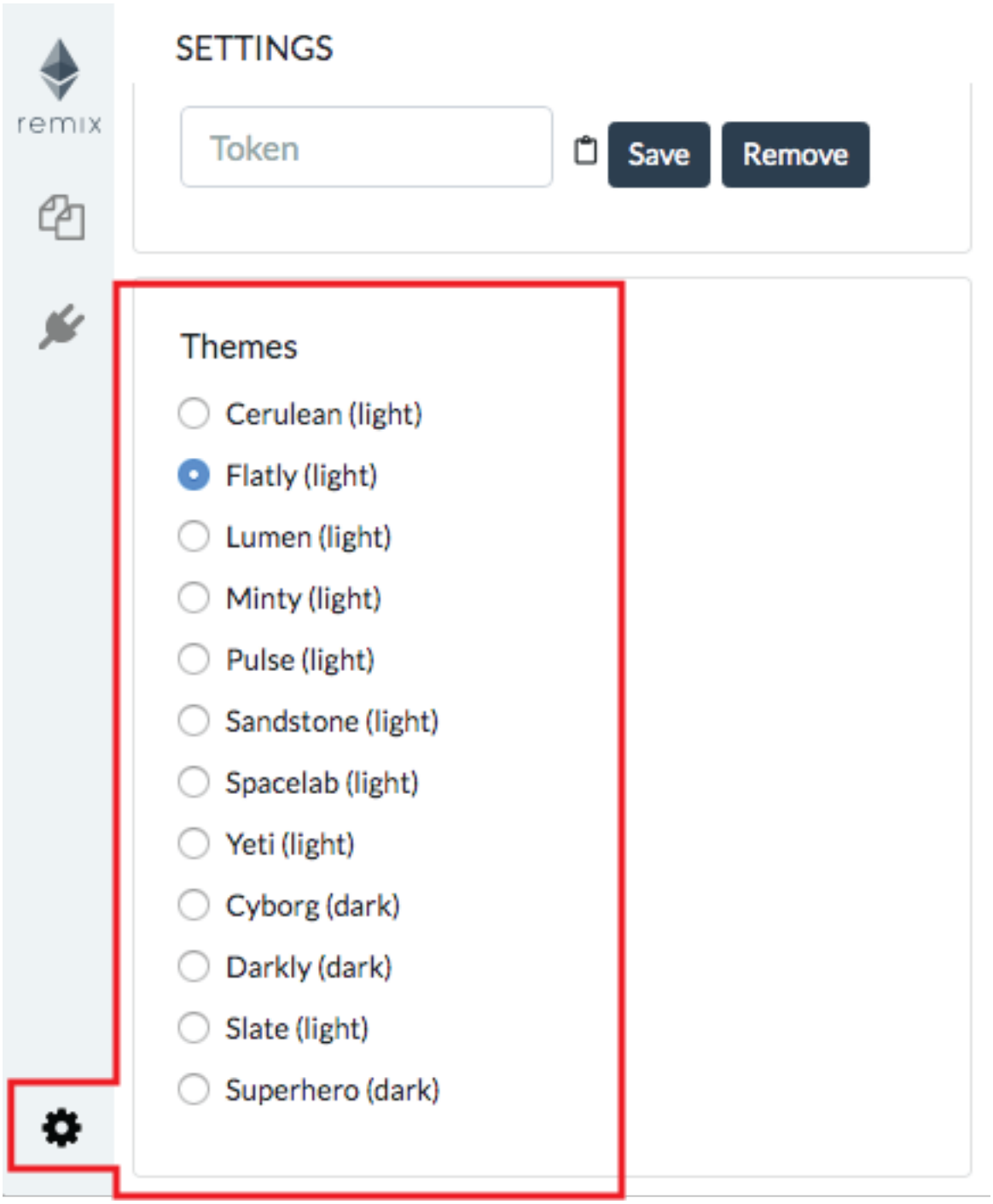
1.4 Plugin Manager

In order to make Remix flexible for integrating changes into its functionality and for integrating remix into other projects (your's for example), we've now made everything a plugin. This means that you only load the functionality you need. It also means that you need a place to turn off and on plugins - as your needs change. This all happens in the plug manager.

The Plugin Manager is also the place you go when you are creating your own plugin and you want to load your local plugin into Remix. In that case you'd click on the "Connect to a Local Plugin" link at the top of the Plugin Manager panel.

1.5 Themes

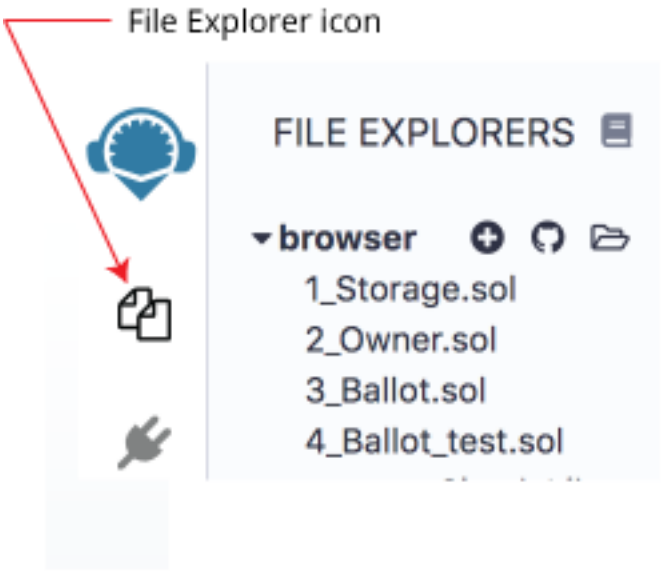
So you want to work on Remix with a dark theme or a gray theme or just a different theme that the one you are currently looking at? Go to the settings tab and at the bottom is a choice of lots of bootstrap based themes.



CHAPTER 2

File Explorers

To get to the File Explorers module - click the file explorers icon.



The basic files explorer lists all the files stored in your browser's **browser storage**. You can see them in the **browser** folder.

Important Note: Clearing the browser storage will **permanently delete** all the solidity files stored there. This is an inherent limitation of a browser-based IDE. However, if you want to store files outside of the browser and on your computer's filesystem, use Remixd or use the desktop version of Remix-IDE. RemixD enables you to have access to a selected folder on your hard drive. Remix Desktop is a version of Remix-IDE in an Electron app.

You can rename, remove or add new files to the file explorer.

We will start by reviewing the icons in the image above.

The book icon - **A.** is the link to the module's documentation.

The icons to the right of the **browser** file explorer in the image above only appear for browser storage.

2.1 Create new File

The icon marked **B.** above. Creates a new file.

2.2 Publish to Gist

The icon marked **C.** above. Publishes all files from the browser folder to a gist. Only file in the root of **browser** will be published. Files in subfolders will not be publish to the Gist. Gist API has changed in 2018 and **requires** users to be authenticated to be able to publish a gist.

Click this link to Github tokens setup and select Generate new token. Then check the **Create gists** checkbox and generate a new token.

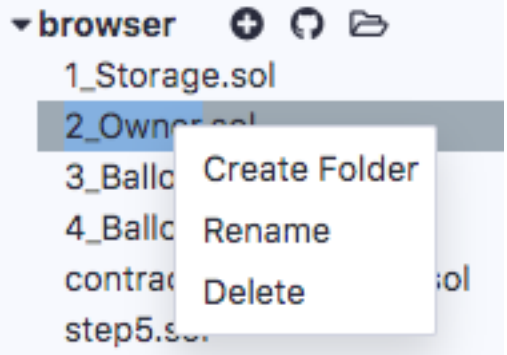
Take the token and paste it in Remix's **Settings** module in the **Github Access Token** section. And then click Save. Now you should be able to use the feature.

2.3 Create a folder

The icon marked **D.** above. Creates a new folder in **browser** file explorer.

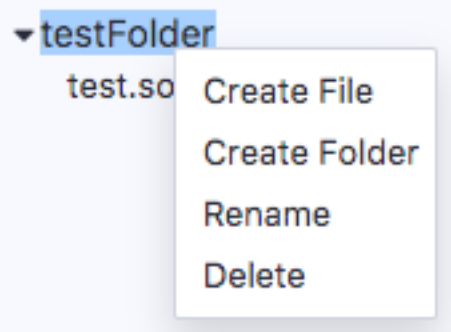
2.4 Context Menu (Right Click)

Right click on a file or a folder and the **context menu** will appear.



You can rename or delete a selected file or a folder. You can also create a folder.

To create a file with the context menu, right click on a folder to get the **Create File** option. A file will be created inside that folder.



The functionality of the context menu also works with RemixD (which gives you have access to a folder on your hard drive).

Note: When working with RemixD, you need to open and close the **localhost** folder to refresh the view.

CHAPTER 3

Plugin Manager

3.1 Everything is a PLUGIN in Remix

In order to integrate new tools made by us and by . . . you into Remix, we've now made everything a plugin. This architecture will also allow Remix or just parts of Remix to be integrated into other projects (your's for example).

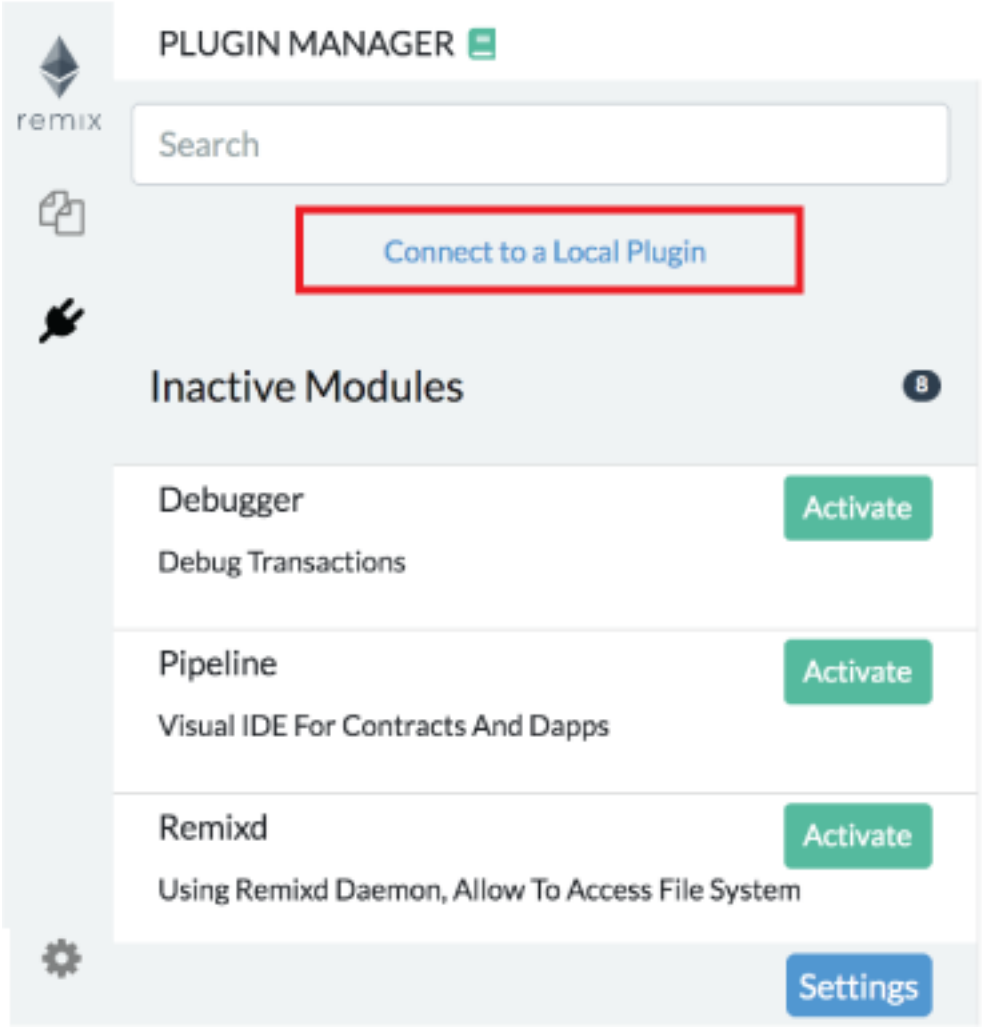
This means that you only load the functionality you need.

It also means that you can turn off and on plugins - as your needs change.

This all happens in the plug manager.

The Plugin Manager is also the place you go when you are creating your own plugin and you want to load your local plugin into Remix.

To load your local plugin, you'd click on the "Connect to a Local Plugin" link at the top of the Plugin Manager panel.



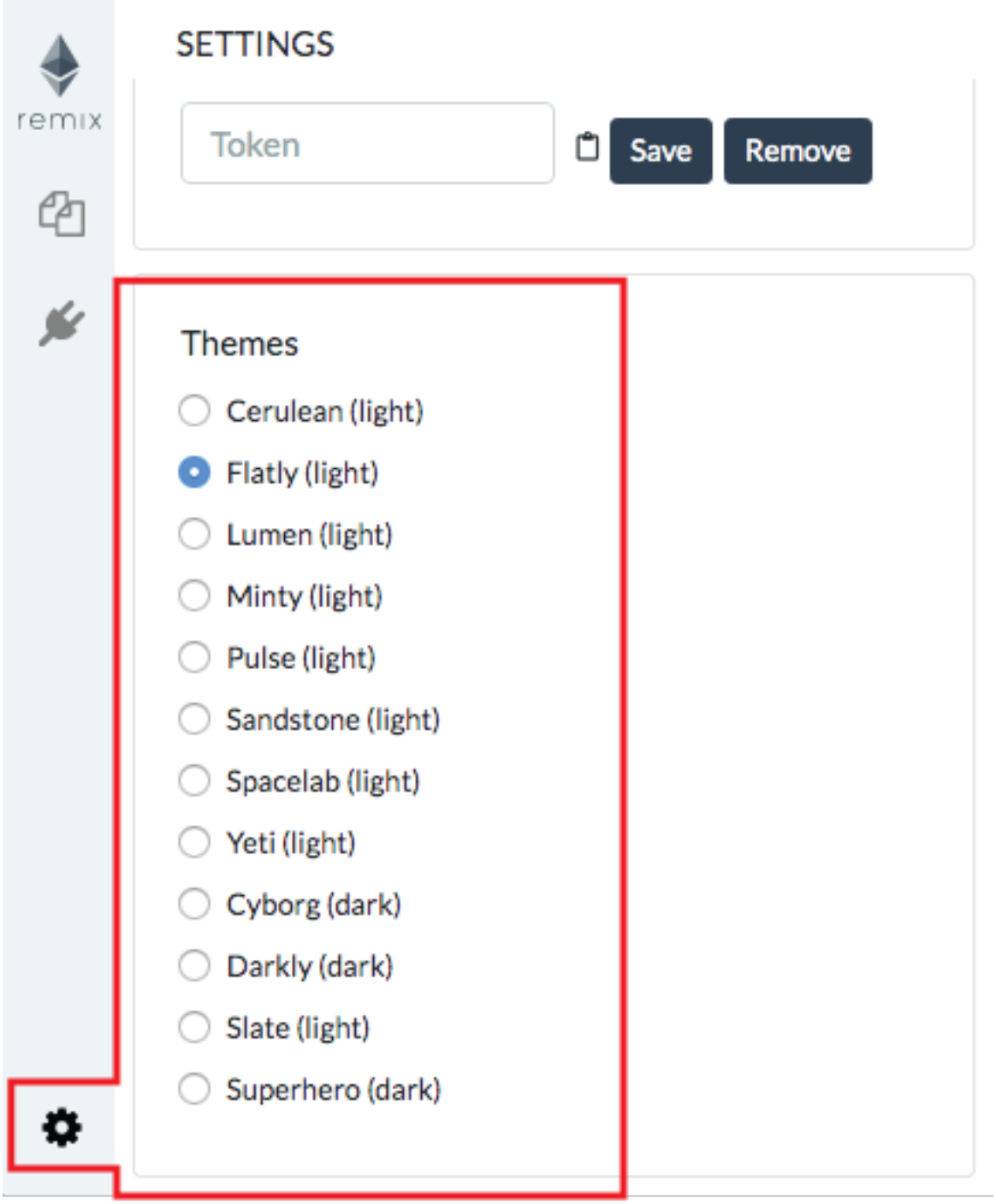
To learn more about how to create your own plugin, go to the README of remix-plugin repo.

CHAPTER 4

Settings

To get to **Settings** click the gear a the very bottom of the icon panel.

You can find a link to the homepage (if you closed it) as well as a link to our Gitter Channel and for you aesthetes out there, we now have a rather large list of themes.



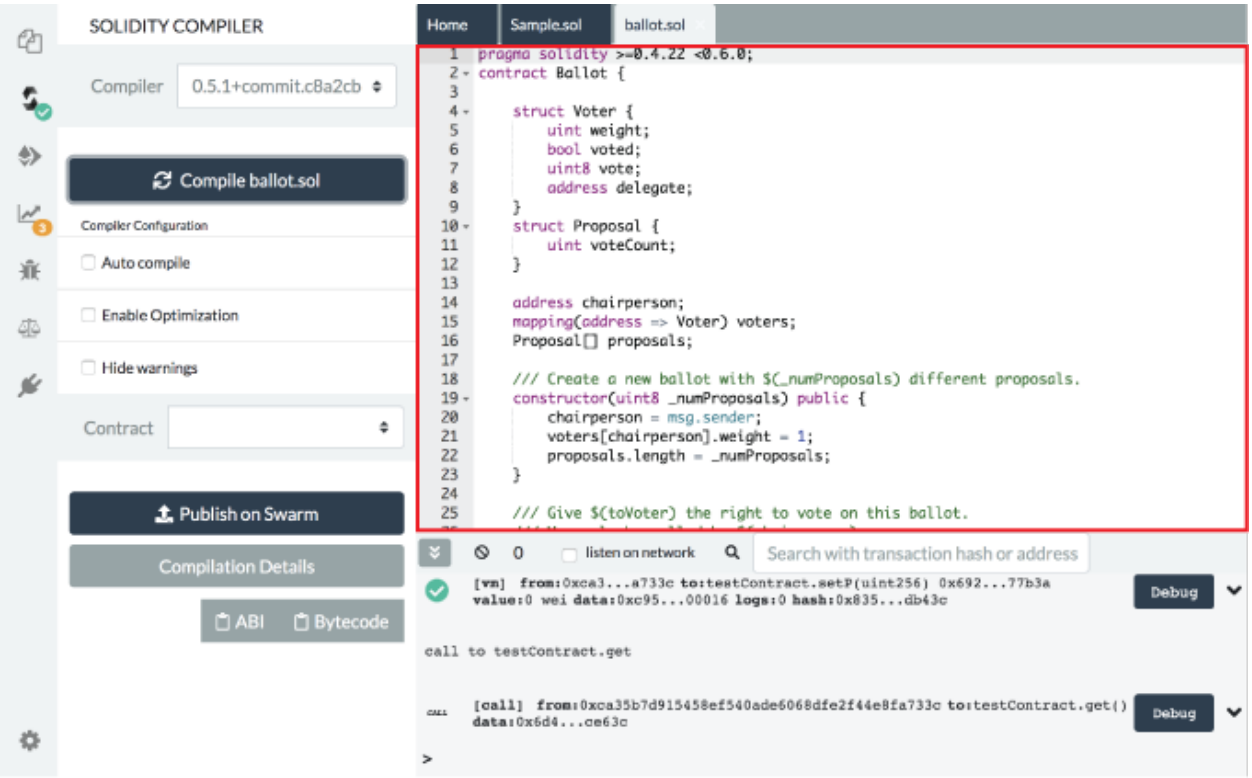
Another important settings:

- Text wrap: controls if the text in the editor should be wrapped.
- Enable optimization: defines if the compiler should enable optimization during compilation. Enabling this option saves execution gas. It is useful to enable optimization for contracts ready to be deployed in production but could lead to some inconsistencies when debugging such a contract.

CHAPTER 5

Solidity Editor

The Remix editor recompiles the code each time the current file is changed or another file is selected. It also provides syntax highlighting mapped to solidity keywords.



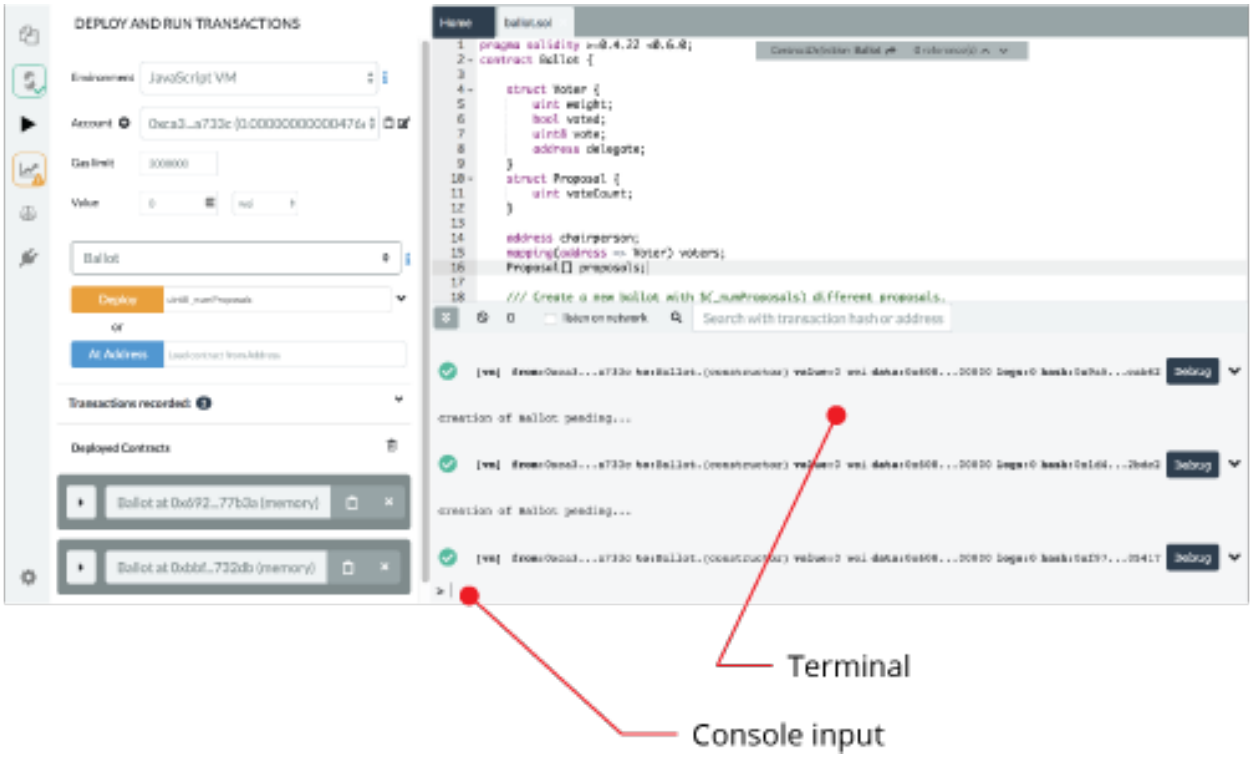
Here's the list of some important features:

- It display opened files as tabs.
- Compilation Warning and Error are displayed in the gutter

- Remix saves the current file continuously (5s after the last changes)
- +/- on the top left corner enable you to increase/decrease the font size of the editor

CHAPTER 6

Terminal



Features, available in the terminal:

- It integrates a JavaScript interpreter and the `web3` object. It enables the execution of the JavaScript script which interacts with the current context. (note that `web3` is only available if the `web provider` or `injected provider` mode is selected).
- It displays important actions made while interacting with the Remix IDE (i.e. sending a new transaction).
- It displays transactions that are mined in the current context. You can choose to display all transactions or only transactions that refers to the contracts Remix knows (e.g transaction created from the Remix IDE).

- It allows searching for the data and clearing the logs from the terminal.
- You can run scripts by inputting them at the bottom after the >.

CHAPTER 7

Compiler (Solidity)

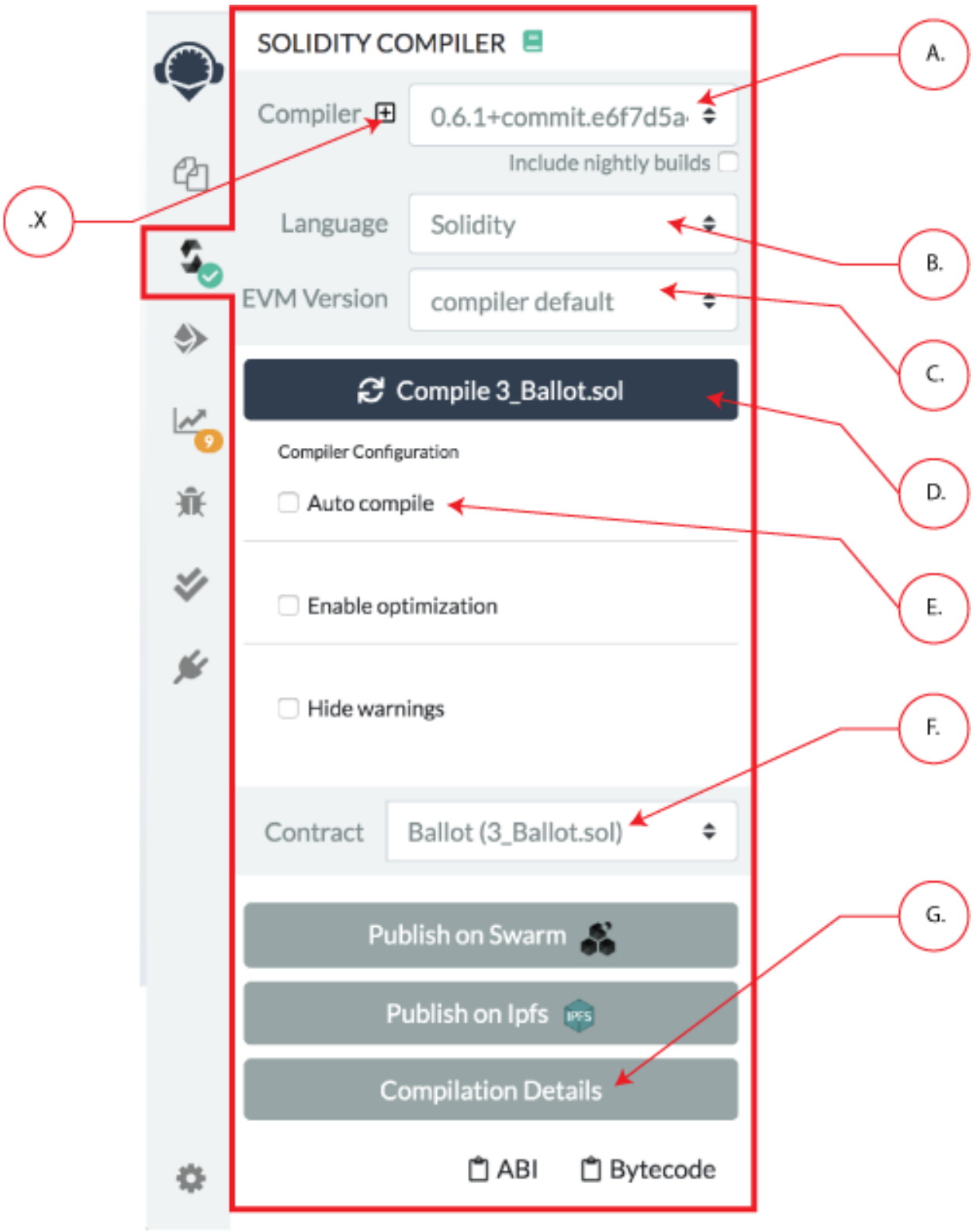
Clicking the Solidity icon in the icon panel brings you to the Solidty Compiler.

Compiling is triggered when you click the compile button (**D. in image below**). If you want the file to be compiled each time the file is saved or when another file is selected - check the auto compile checkbox (**E. in image below**).

Since the Solidity version `0.5.7`, it is possible to compile `Yul` files. Please read the (solidity documentation about Yul) which contain some code examples. You can use the language dropdown (**B. in image below**) to switch the language. **This dropdown list is only available for versions greater than or equal to `0.5.7`.**

The fork selection dropdown list (**C. in image below**) allows to compile code against a specific ethereum hard fork. The `compiler default` corresponds to the default hard fork used by a specific version. Please go to "Compilation Details" (**G. in image below**) in the settings of `Metadata` section to see the harfork name used for the current compilation.

If the contract has a lot of dependencies it can take a while to compile - so you use autocompilation at your discretion.



After each compilation, a list is updated with all newly compiled contracts. A compiled contract can be selected with the Contract pulldown menu (**F. in the image**). Multiple contracts are compiled when one contract imports other contracts. Selecting a contract will show information about that one.

When the "Compilation Details" button is clicked (**G. in image**), a modal opens displaying detailed information about the current selected contract.

For those writing your own custom solidity compiler, you can import that by clicking the **+** button (**X. in the image**) to open a modal where you can input the url of the compiler to be loaded.

From the Solidity Compiler module you can also publish your contract to Swarm (only non abstract contracts can be published) & IPFS.

Published data notably contains the `abi` and the solidity source code.

After a contract is published, you can find its metadata information using the bzz URL located in the details modal dialog `SWARM LOCATION`.

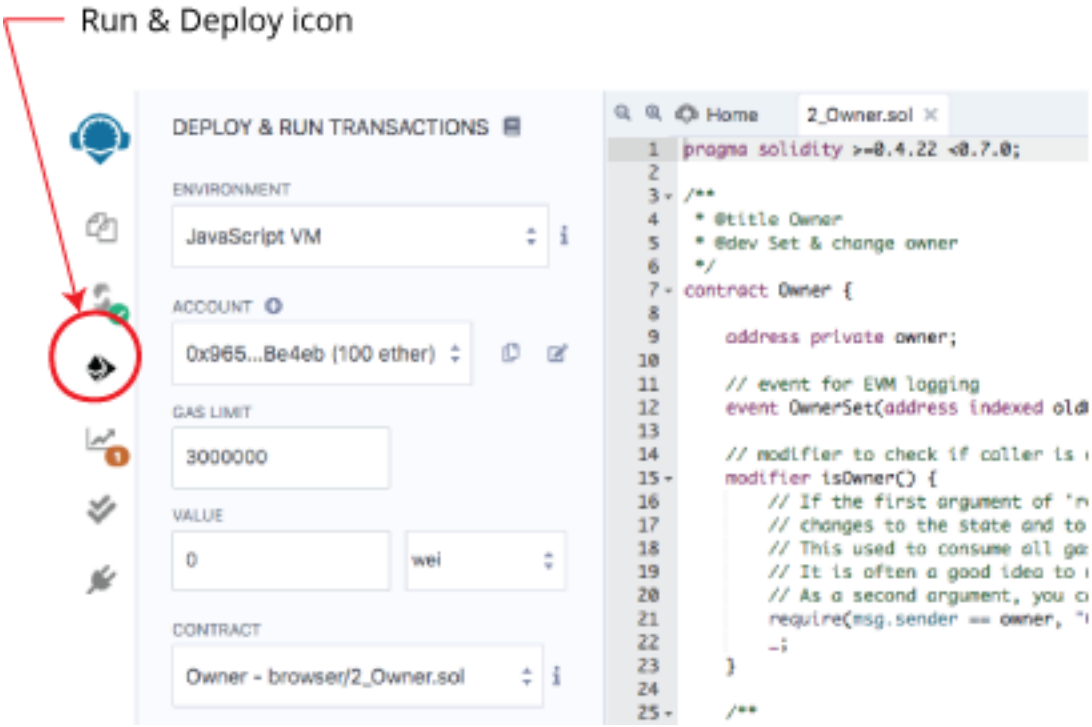
Compilation Errors and Warning are displayed below the contract section. At each compilation, the static analysis tab builds a report. It is important to address reported issues even if the compiler doesn't complain. (see more)

CHAPTER 8

Deploy & Run

The Deploy & Run module allows you to send transactions to the current environment.

To use this module, you need to have a contract compiled. So, if there is a contract name in the CONTRACT select box (the select box is under the VALUE input field), you can use this module. If nothing is there or you do not see the contract you want, you need to select a contract in the editor to make it active, go to a compiler module and compile it, and then come back to Deploy & Run.



8.1 Environment

- `JavaScript VM`: All the transactions will be executed in a sandbox blockchain in the browser. This means nothing will be persisted when you reload the page. The JsVM is its own blockchain and on each reload it will start a new blockchain, the old one will not be saved.
- `Injected Provider`: Remix will connect to an injected web3 provider. `Metamask` is an example of a provider that inject web3.
- `Web3 Provider`: Remix will connect to a remote node. You will need to provide the URL to the selected provider: geth, parity or any Ethereum client.

8.2 More about Web3 Provider

If you are using Geth & https://remix.ethereum.org, please use the following Geth command to allow requests from Remix:

geth –rpc –rpccorsdomain https://remix.ethereum.org

Also see Geth Docs about the rpc server

To run Remix using https://remix.ethereum.org & a local test node, use this Geth command:

geth –rpc –rpccorsdomain=”https://remix.ethereum.org” –rpcapi web3,eth,debug,personal,net –vmdebug –datadir <path/to/local/folder/for/test/chain> –dev console

If you are using remix-alpha or a local version of remix - replace the url of the –rpccorsdomain with the url of Remix that you are using.

To run Remix Desktop & a local test node, use this Geth command:

geth –rpc –rpccorsdomain=”package://a7df6d3c223593f3550b35e90d7b0b1f.mod” –rpcapi web3,eth,debug,personal,net –vmdebug –datadir <path/to/local/folder/for/test/chain> –dev console

Also see Geth Docs on Dev mode

The Web3 Provider Endpoint for a local node is **http://localhost:8545**

WARNING: Don’t get lazy. It is a bad idea to use the Geth flag –rpccorsdomain with a wildcard: `--rpccorsdomain *`

If you put the wildcard `*`, it means everyone can request the node. Whereas, if you put a URL, it restricts the urls to just that one - e.g. `--rpccorsdomain 'https://remix-alpha.ethereum.org'`

Only use `--rpccorsdomain *` when using a **test chain** AND using only **test accounts**. For real accounts or on the mainchain **specify the url**.

8.3 Account:

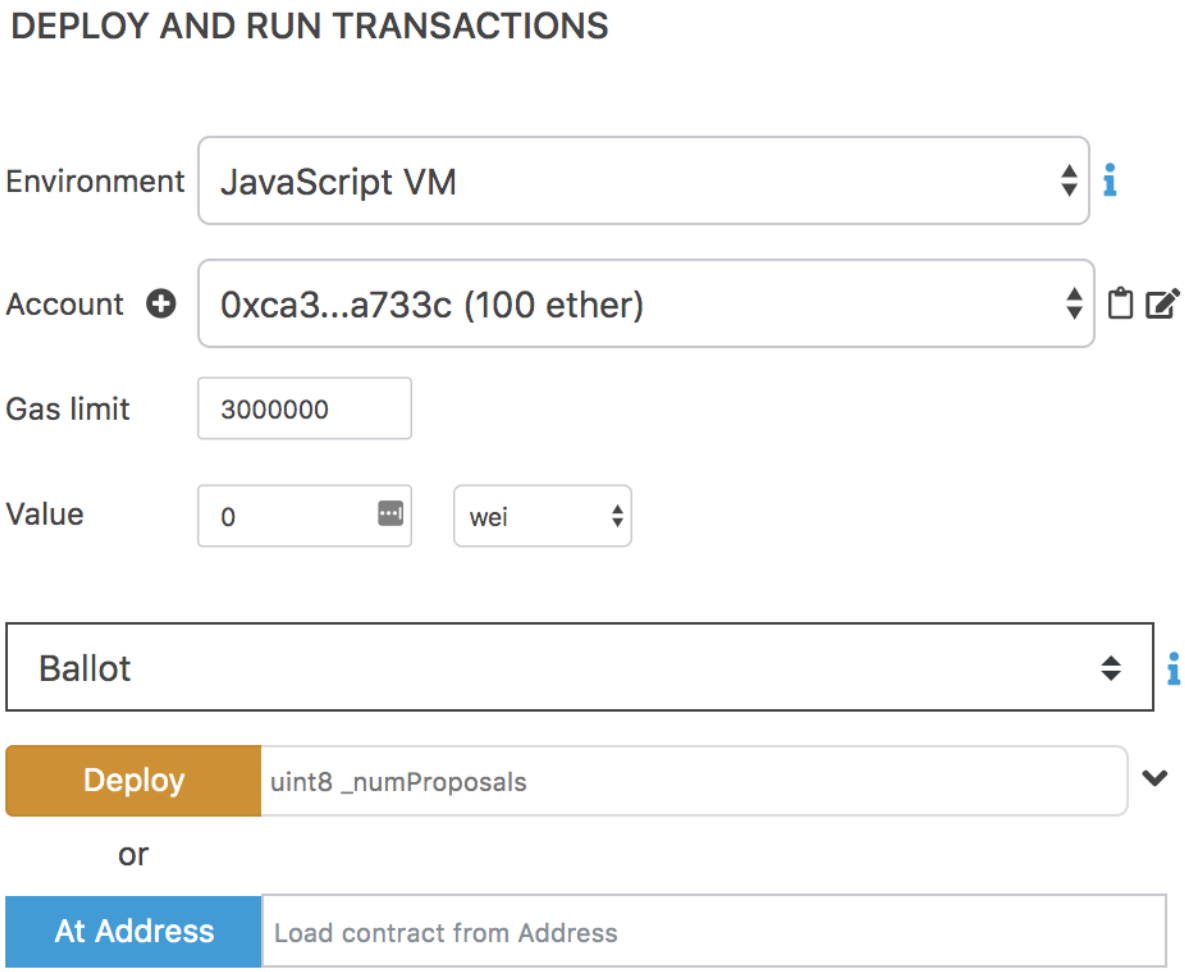
- Account: the list of accounts associated with the current environment (and their associated balances). On the JsVM, you have a choice of 5 accounts. If using Injected Web3 with MetaMask, you need to change the account in MetaMask.

8.4 Gas Limit:

- This sets the maximum amount of gas that will be allowed for all the transactions created in Remix.

8.5 Value:

- This sets the amount of ETH, WEI, GWEI etc that is sent to a contract or a payable function. (Note: payable functions have a red button). The value is always reset to 0 after each transaction execution). The Value field is **NOT** for gas.



8.6 Initiate Instance

- In the image above, the select box is set to Ballot. This select box will contain the list of compiled contracts.
- `Deploy` send a transaction that deploys the selected contract. When the transaction is mined, the newly created instance will be added (this might take several seconds). Note that if the `constructor` has parameters, you need to specify them.

- `At Address` this is used at access a contract that has already been deployed. It assumes that the given address is an instance of the selected contract. **Note:** There's no check at this point, so be careful when using this feature, and be sure you trust the contract at that address.

8.7 Pending Instances

Validating a transaction takes several seconds. During this time, the GUI shows it in a pending mode. When the transaction is mined, the number of pending transactions is updated and the transaction is added to the log (see terminal).

8.8 Using the ABI

Using `Deploy` or `At Address` is a classic use case of Remix. However, it is possible to interact with a contract by using its ABI. The ABI is a JSON array which describe its interface.

To interact with a contract using the ABI, create a new file in Remix with extension `*.abi` and copy the ABI content to it. Then, in the input next to `At Address`, put the Address of the contract you want to interact with. Click on `At Address`, a new "connection" with the contract will popup below.

8.9 Using the Recorder

The Recorder is a tool used to save a bunch of transactions in a JSON file and rerun them later either in the same environment or in another.

Saving to the JSON file (by default its called scenario.json) allows one to easily check the transaction list, tweak input parameters, change linked library, etc. . .

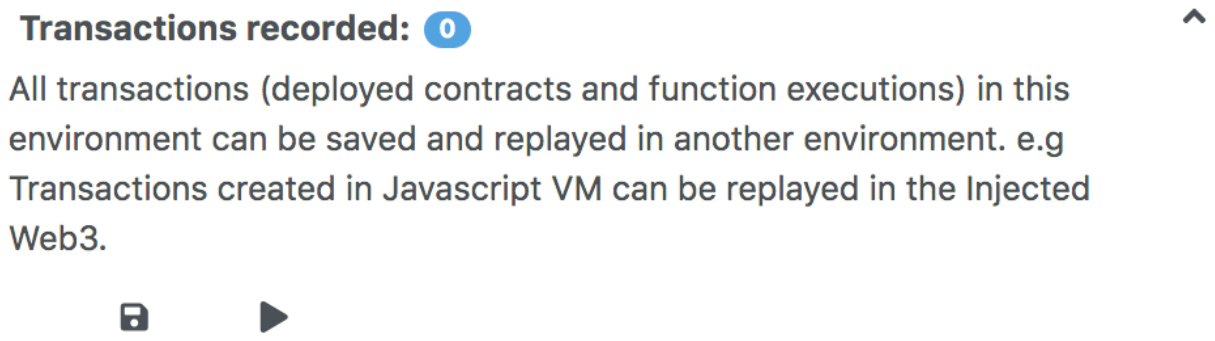
There are many use cases for the recorder.

For instance:

- After having coded and tested contracts in a constrained environment (like the JavaScript VM), you could then change the environment and redeploy it to a more realistic environment like a test net with an **injected web3** or to a Geth node. By using the generated **scenario.json** file, you will be using all the same settings that you used in the Javascript VM. And this mean that you won't need to click the interface 100 times or whatever to get the state that you achieved originally. So the recorder could be a tool to protect your sanity.

  You can also change the settings in the scenario.json file to customize the playback.

- Deploying contract does often require more than creating one transaction and so the recorder will automate this deployment.
- Working in a dev environment often requires to setup the state in a first place.



8.9.1 scenario.json

To create this file in the recorder, you first of course need to have run some transactions. In the image above - it has a `0` next to **Transactions Recorded**. So this isn't the right moment to save transactions because - well because there aren't any. Each time you make a transaction, that number will increment. Then when you are ready, click the floppy disk icon and the scenario.json file will be created.

The JSON file below is an example of the scenario.json file.

In it, 3 transactions are executed:

The first corresponds to the deployment of the lib `testLib`.

The second corresponds to the deployment of the contract `test` with the first parameter of the constructor set to 11. That contract depends on a library. The linkage is done using the property `linkReferences`. In that case we use the address of the previously created library : `created{1512830014773}`. The number is the id (timestamp) of the transaction that led to the creation of the library.

The third record corresponds to the call to the function `set` of the contract `test` (the property to is set to: `created{1512830015080}`) . Input parameters are `1` and `0xca35b7d915458ef540ade6068dfe2f44e8fa733c`

All these transactions are created using the value of the accounts `account{0}`.

```
{
"accounts": {
    "account{0}": "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
},
"linkReferences": {
    "testLib": "created{1512830014773}"
},
"transactions": [
    {
    "timestamp": 1512830014773,
    "record": {
        "value": "0",
        "parameters": [],
        "abi": "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a",
        "contractName": "testLib",
        "bytecode":
↪"60606040523415600e57600080fd5b60968061001c6000396000f300606060405260043610603f576000357c0100000000
↪",
        "linkReferences": {},
        "type": "constructor",
```

(continues on next page)

(continued from previous page)


```
        "from": "account{0}"
    }
    },
    {
    "timestamp": 1512830015080,
    "record": {
        "value": "100",
        "parameters": [
        11
        ],
        "abi": "0xc41589e7559804ea4a2080dad19d876a024ccb05117835447d72ce08c1d020ec",
        "contractName": "test",
        "bytecode":
↪"606060405260405160208061102b1833981016040528080519060200190919050508060008190555050610 27a806100376
↪_browser/ballot.sol:testLib____________
↪636d4ce63c60006040516020015260405181 63ffffffff167c0100000000000000000000000000000000000000000000000
↪",
        "linkReferences": {
        "browser/ballot.sol": {
            "testLib": [
            {
                "length": 20,
                "start": 511
            }
            ]
        }
        },
        "name": "",
        "type": "constructor",
        "from": "account{0}"
    }
    },
    {
    "timestamp": 1512830034180,
    "record": {
        "value": "1000000000000000000",
        "parameters": [
        1,
        "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
        ],
        "to": "created{1512830015080}",
        "abi": "0xc41589e7559804ea4a2080dad19d876a024ccb05117835447d72ce08c1d020ec",
        "name": "set",
        "type": "function",
        "from": "account{0}"
    }
    }
],
"abis": {
    "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a": [
    {
        "constant": true,
        "inputs": [],
        "name": "get",
        "outputs": [
        {
            "name": "",
```


(continues on next page)

(continued from previous page)


```
            "type": "uint256"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    }
    ],
    "0xc41589e7559804ea4a2080dad19d876a024ccb05117835447d72ce08c1d020ec": [
    {
        "constant": true,
        "inputs": [],
        "name": "getInt",
        "outputs": [
        {
            "name": "",
            "type": "uint256"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "getFromLib",
        "outputs": [
        {
            "name": "",
            "type": "uint256"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "getAddress",
        "outputs": [
        {
            "name": "",
            "type": "address"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
        {
            "name": "_t",
            "type": "uint256"
```


(continues on next page)

(continued from previous page)

```
        },
        {
            "name": "_add",
            "type": "address"
        }
        ],
        "name": "set",
        "outputs": [],
        "payable": true,
        "stateMutability": "payable",
        "type": "function"
    },
    {
        "inputs": [
        {
            "name": "_r",
            "type": "uint256"
        }
        ],
        "payable": true,
        "stateMutability": "payable",
        "type": "constructor"
    }
    ]
}
}
```

CHAPTER 9

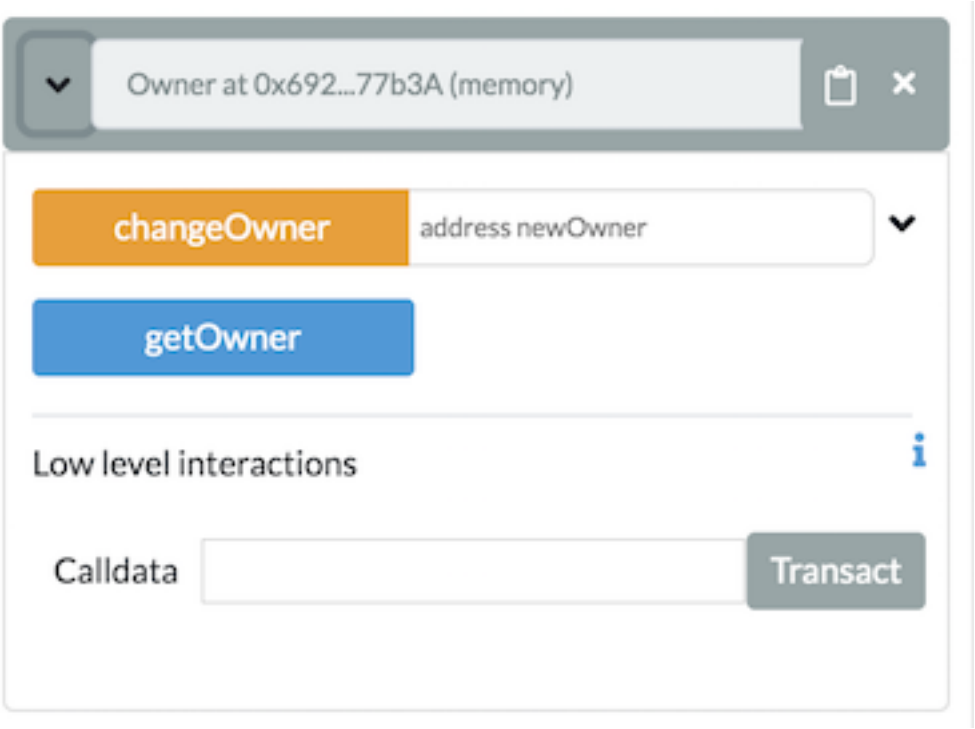
Run & Deploy (part 2)

9.1 Deployed contracts

This section in the Run tab contains a list of deployed contracts to interact with through autogenerated UI of the deployed contract (also called udapp).

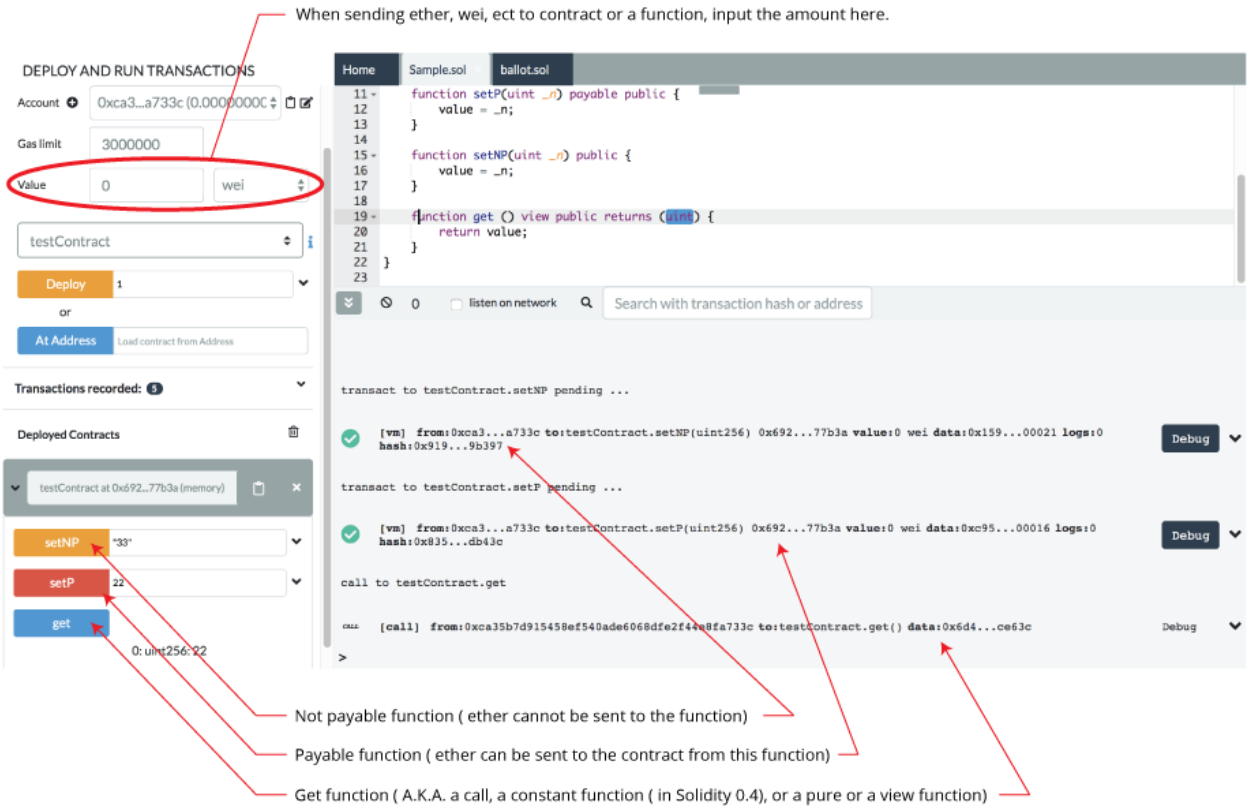
The deployed contract appears but is in its collapsed form.

Click the sideways caret to open it up.



You will see the functions in the contract. The functions buttons can have different color buttons.

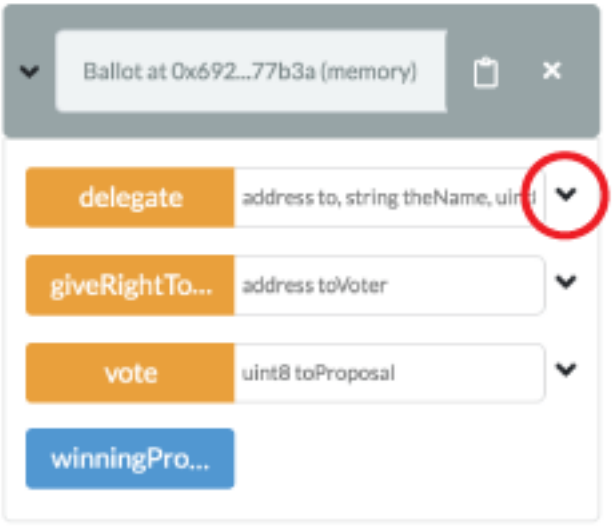
- Functions that are `constant` or `pure` functions in Solidity have a blue buttons. Clicking one of this type does not create a new transaction. So clicking will not cause state changes - it will only return a value stored in the contract - so it won't cost you anything in gas fees.
- Functions that change the state of the contract AND that do not accept Ether are called `non-payable` functions and have an orange button. Clicking on them will create a transaction and thus cost gas.
- Functions that have red buttons are `payable` functions in Solidity. Clicking one of these will create a new transaction and this transaction can accept a **value**. The **value** is put in in the Value field which is under the Gas Limit field.



See more information about Solidity modifiers in the Solidity docs. .

If a function requires input parameters, well.. you gotta put them in.

9.2 Inputting parameters



9.2.1 Inputting parameters in the collapsed view

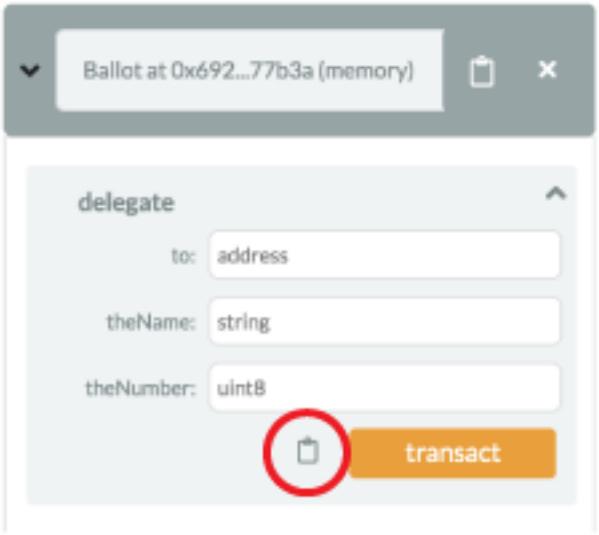
(Inputting all the parameters in a single input box)

- The input box tells you what type each parameter needs to be.
- Numbers and addresses do not need to be wrapped in double quotes.
- Strings need to be wrapped.
- Parameters are separated by commas.

In the example above the "delegate" function has 3 parameters.

9.2.2 Inputting parameters in the expanded view

Clicking the 'down' caret brings you to the *Multi-param Manager* - where you can input the parameters one at a time. **Much less confusing!**



In the expanded view, strings do not need to be wrapped.

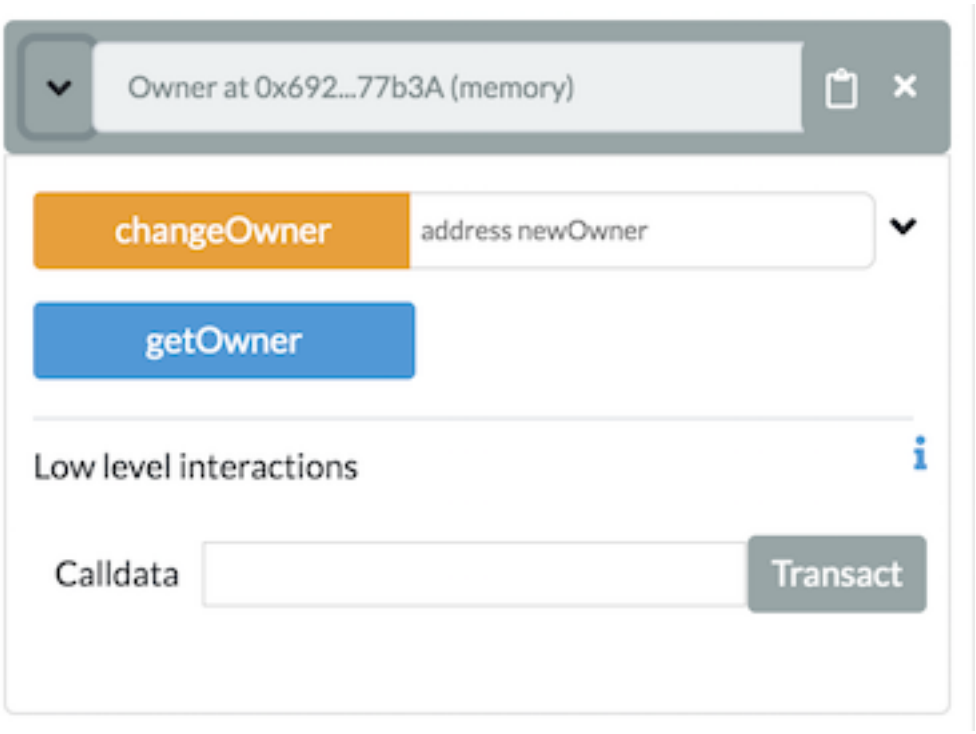
Clicking the clipboard icon will encode the inputs and will copy them. Only a valid set of inputs can be encoded.

So if you made a mistake and put a uint8 where an address should have been, clicking the clipboard here will give you an error.

9.3 Low level interactions

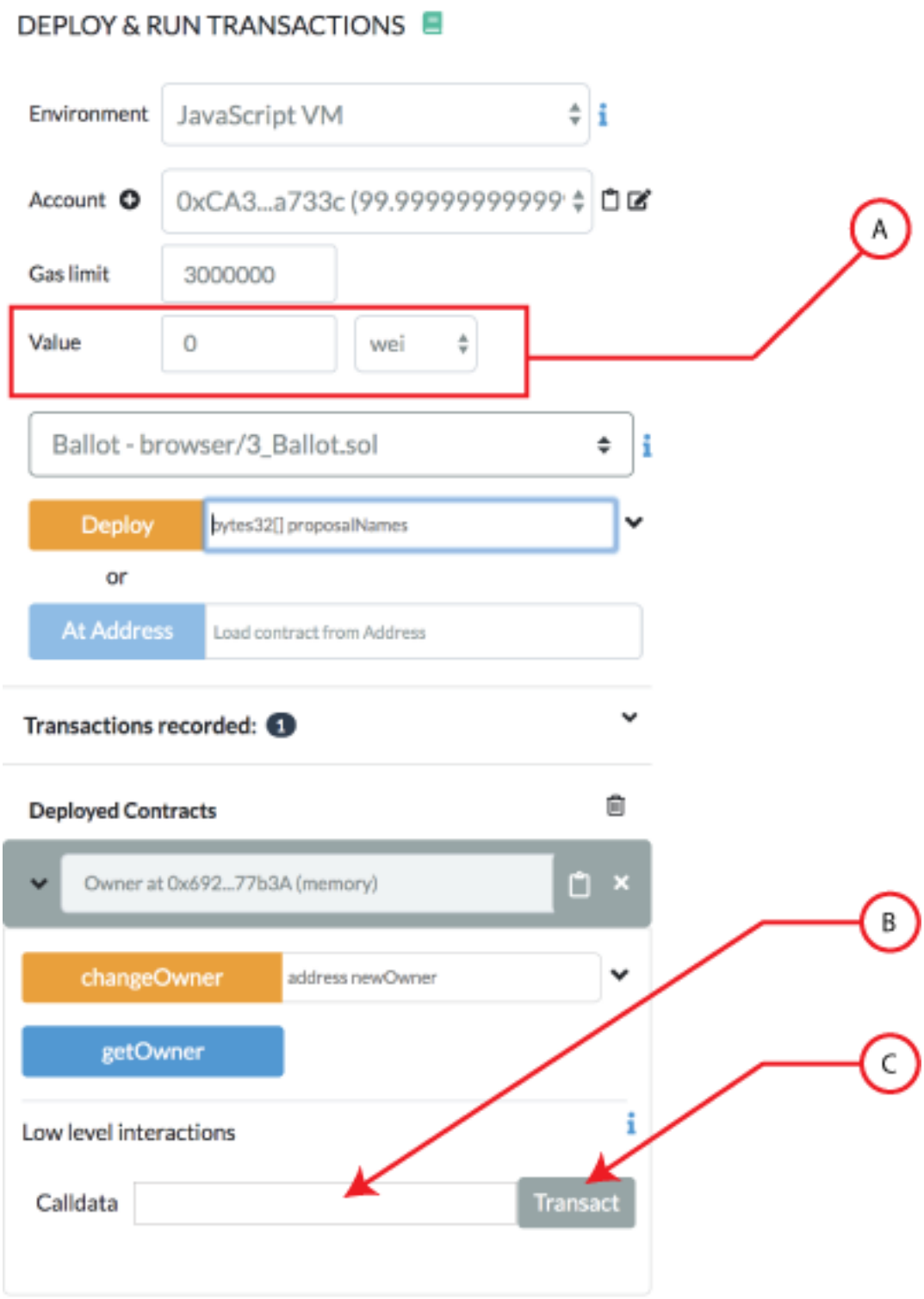
Low level interactions are used to send funds or calldata or funds & calldata to a contract through the **recieve()** or **fallback()** function. Typically, you should only need to implement the fallback function if you are following an upgrade or proxy pattern.

The low level interactions section is below the functions in each deployed contract.



Please note the following:

- If you are executing a plain Ether transfer to a contract, you need to have the receive() function in your contract. If your contract has been deployed and you want to send it funds, you would input the amount of Ether or Wei etc. (see **A** in graphic below), and then input **NOTHING** in the calldata field of **Low level interactions** (see **B** in graphic) and click the Transact button (see **C** in graphic below).



- If you are sending calldata to your contract with Ether, then you need to use the fallback() function and have it with the state mutability of **payable**.
- If you are not sending ether to the contract but **are** sending call data then you need to use the fallback() function.
- If you break the rules when using the **Low level interactions** you will be slapped with a warning.

Please see the solidity docs for more specifics about using the **fallback** and **receive** functions.

9.3.1 Passing in a tuple or a struct to a function

To pass a tuple in, you need to put in an array [].

Similarly, to pass in a struct as a parameter of a function, it needs to be put in as an array []. Also note that the line `pragma experimental ABIEncoderV2;` needs to put in at the top of the solidity file.

9.3.2 Example of passing nested struct to a function

Consider a nested struct defined like this:

```
struct gardenPlot {
    uint slugCount;
    uint wormCount;
    Flower[] theFlowers;
}
struct Flower {
    uint flowerNum;
    string color;
}
```

If a function has the signature `fertilizer(Garden memory gardenPlot)` then the correct syntax is:

```
[1,2,[[3,"Petunia"]]]
```

To continue on this example, here's a sample contract:

```
pragma solidity >=0.4.22 <0.7.0;
pragma experimental ABIEncoderV2;

contract Sunshine {
    struct Garden {
      uint slugCount;
      uint wormCount;
      Flower[] theFlowers;
    }
    struct Flower {
        uint flowerNum;
        string color;
    }

    function picker(Garden memory gardenPlot) public {
        uint a = gardenPlot.slugCount;
        uint b = gardenPlot.wormCount;
        Flower[] memory cFlowers = gardenPlot.theFlowers;
        uint d = gardenPlot.theFlowers[0].flowerNum;
        string memory e = gardenPlot.theFlowers[0].color;
    }
}
```

After compiling, deploying the contract and opening up the deployed instance, we can then add the following input parameters to the function named **fertilizer** :

```
[1,2,[[3,"Black-eyed Susan"],[4,"Pansy"]]]
```

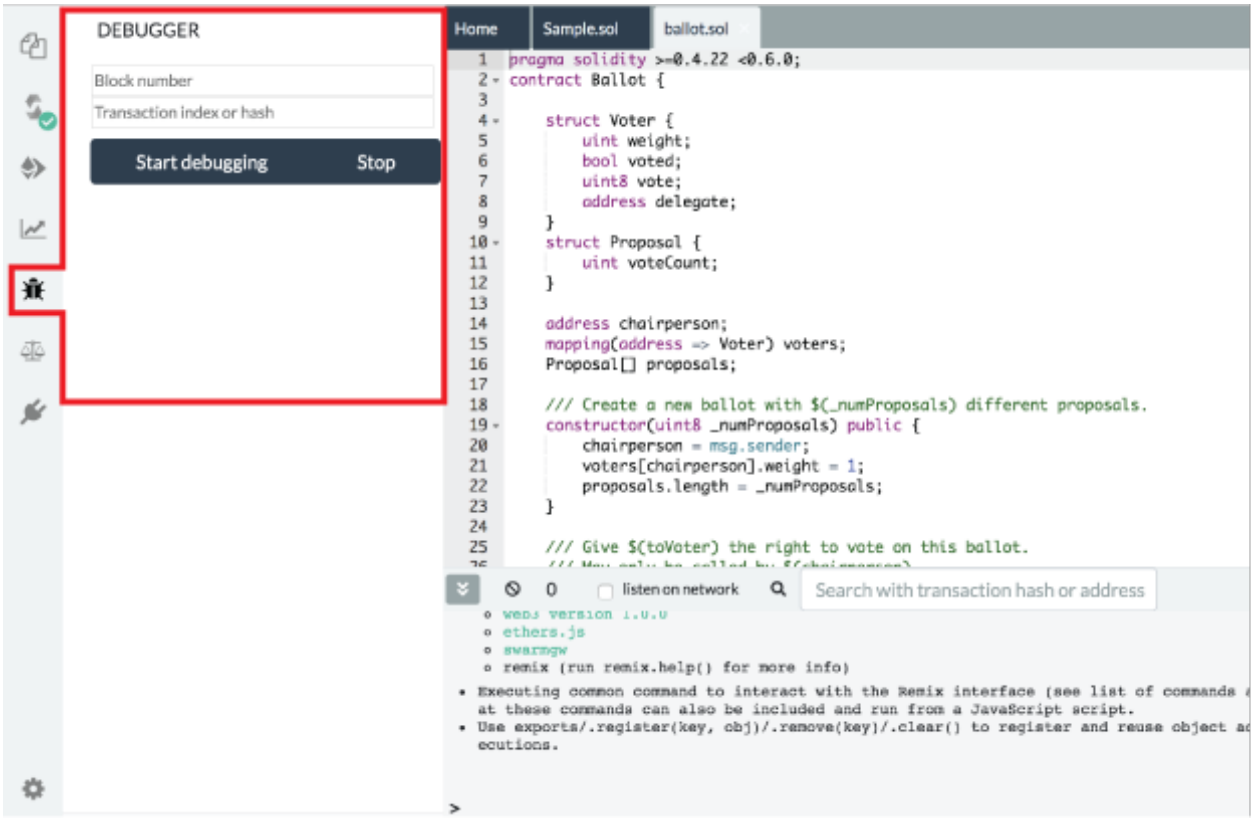
The function **fertilizer** accepts a single parameter of the type **Garden**. The type **Garden** is a **struct**. Structs are wrapped in **square brackets**. Inside **Garden** is an array that is an array of structs named **theFlowers**. It gets a set of brackets for the array and another set for the struct. Thus the double square brackets.

CHAPTER 10

Debugger

This module allows you to debug the transaction. It can be used to deploy transactions created from Remix and already mined transactions. (debugging works only if the current environment provides the necessary features).

To get to the debugger - you can click the debug button in the terminal when a successful or failed transaction appears there. You can also load the module from the plugin manager and then click the bug in the icon panel. Or you can get to the debugger by running the debug command in the console.



To learn more about how to use this tool go to the debugger tutorial.

CHAPTER 11

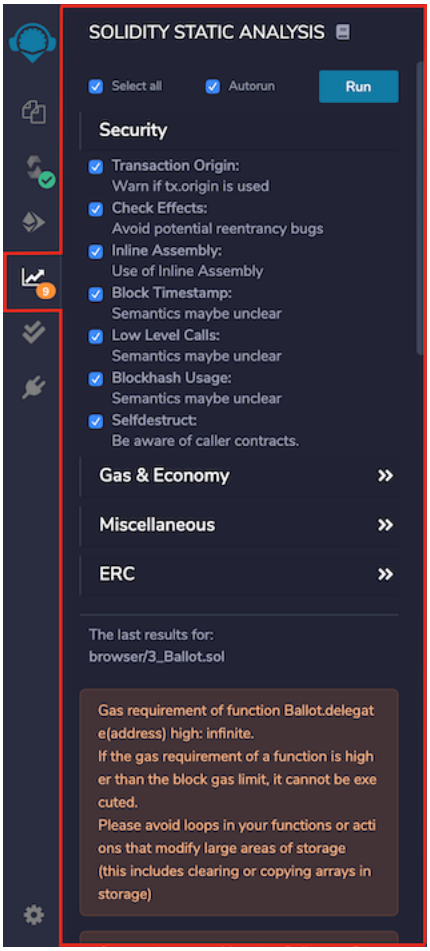
Solidity Static Analysis

Static code analysis is a process to debug the code by examining it and without actually executing the code.

`Solidity Static Analysis` plugin performs static analysis on Solidity smart contracts once they are compiled. It checks for security vulnerabilities and bad development practices, among other issues. This plugin comes with `Solidity` environment of Remix IDE. It can also be activated individually from `Plugin Manager`.

11.1 How to use

If you select this plugin, you will see a number of modules listed along with checkboxes, one `Auto run` checkbox and a `Run` button.



By default, all modules are selected for analysis and a new analysis is performed at each compilation.

One can select/deselect the modules under which contract should be analyzed and can run the analysis again for last compiled contract by clicking on `Run`.

If you don't want to run analysis each time you compile a contract, just uncheck the checkbox near to `Auto run`.

11.2 Analysis Modules

Currently, with Remix IDE v0.10.1, there are 21 analysis modules listed under 4 categories. Categories are: `Security`, `Gas & Economy`, `ERC` & `Miscellaneous`.

Here is the list of modules under each category along with the example code which **should be avoided or used very carefully while development**:

11.2.1 Category: Security

- **Transaction origin: 'tx.origin' is used**

`tx.origin` is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.

Example:

```
require(tx.origin == owner);
```

- **Check effects: Potential reentrancy bugs**

Potential Violation of Checks-Effects-Interaction pattern can lead to re-entrancy vulnerability.

Example:

```
// sending ether first
msg.sender.transfer(amount);

// updating state afterwards
balances[msg.sender] -= amount;
```

- **Inline assembly: Inline assembly used**

Use of inline assembly is advised only in rare cases.

Example:

```
assembly {
            // retrieve the size of the code, this needs assembly
            let size := extcodesize(_addr)
}
```

- **Block timestamp: Semantics maybe unclear**

`now` does not mean current time. `now` is an alias for `block.timestamp`. `block.timestamp` can be influenced by miners to a certain degree, be careful.

Example:

```
// using now for date comparison
if(startDate > now)
    isStarted = true;

// using block.timestamp
uint c = block.timestamp;
```

- **Low level calls: Semantics maybe unclear**

Use of low level `call`, `callcode` or `delegatecall` should be avoided whenever possible. `send` does not throw an exception when not successful, make sure you deal with the failure case accordingly. Use `transfer` whenever failure of the ether transfer should rollback the whole transaction.

Example:

```
x.call('something');
x.send(1 wei);
```

- **Blockhash usage: Semantics maybe unclear**

`blockhash` is used to access the last 256 block hashes. A miner computes the block hash by "summing up" the information in the current block mined. By summing up the information in a clever way a miner can try to influence the outcome of a transaction in the current block.

Example:

```
bytes32 b = blockhash(100);
```

- **Selfdestruct: Beware of caller contracts**

`selfdestruct` can block calling contracts unexpectedly. Be especially careful if this contract is planned to be used by other contracts (i.e. library contracts, interactions). Selfdestruction of the callee contract can leave callers in an inoperable state.

Example:

```
selfdestruct(address(0x123abc..));
```

11.2.2 Category: Gas & Economy

- **Gas costs: Too high gas requirement of functions**

If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage

Example:

```
for (uint8 proposal = 0; proposal < proposals.length; proposal++) {
    if (proposals[proposal].voteCount > winningVoteCount) {
        winningVoteCount = proposals[proposal].voteCount;
        winningProposal = proposal;
    }
}
```

- **This on local calls: Invocation of local functions via 'this'**

Never use `this` to call functions in the same contract, it only consumes more gas than normal local calls.

Example:

```
contract test {

    function callb() public {
        address x;
        this.b(x);
    }

    function b(address a) public returns (bool) {}
}
```

- **Delete on dynamic Array: Use require/assert appropriately**

The `delete` operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

Example:

```
contract arr {
    uint[] users;
    function resetState() public{
        delete users;
    }
}
```

- **For loop over dynamic array: Iterations depend on dynamic array's size**

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations

in a loop can grow beyond the block gas limit which can stall the complete contract at a certain point. Additionally, using unbounded loops can incur in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

Example:

```
contract forLoopArr {
    uint[] array;

    function shiftArrItem(uint index) public returns(uint[] memory) {
        for (uint i = index; i < array.length; i++) {
            array[i] = array[i+1];
        }
        return array;
    }
}
```

- **Ether transfer in loop: Transferring Ether in a for/while/do-while loop**

Ether payout should not be done in a loop. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. If required, make sure that number of iterations are low and you trust each address involved.

Example:

```
contract etherTransferInLoop {
    address payable owner;

    function transferInForLoop(uint index) public  {
        for (uint i = index; i < 100; i++) {
            owner.transfer(i);
        }
    }

    function transferInWhileLoop(uint index) public  {
        uint i = index;
        while (i < 100) {
            owner.transfer(i);
            i++;
        }
    }
}
```

11.2.3 Category: ERC

- **ERC20: 'decimals' should be 'uint8'**

ERC20 Contracts `decimals` function should have `uint8` as return type.

Example:

```
contract EIP20 {

    uint public decimals = 12;
}
```

11.2.4 Category: Miscellaneous

- **Constant/View/Pure functions: Potentially constant/view/pure functions**

It warns for the methods which potentially should be constant/view/pure but are not.

Example:

```
function b(address a) public returns (bool) {
        return true;
}
```

- **Similar variable names: Variable names are too similar**

It warns on the usage of similar variable names.

Example:

```
// Variables have very similar names voter and voters.
function giveRightToVote(address voter) public {
    require(voters[voter].weight == 0);
    voters[voter].weight = 1;
}
```

- **No return: Function with 'returns' not returning**

It warns for the methods which define a return type but never explicitly return a value.

Example:

```
function noreturn(string memory _dna) public returns (bool) {
       dna = _dna;
   }
```

- **Guard conditions: Use 'require' and 'assert' appropriately**

Use `assert(x)` if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use `require(x)` if x can be false, due to e.g. invalid input or a failing external component.

Example:

```
assert(a.balance == 0);
```

- **Result not used: The result of an operation not used**

A binary operation yields a value that is not used in the following. This is often caused by confusing assignment (=) and comparison (==).

Example:

```
c == 5;
or
a + b;
```

- **String Length: Bytes length != String length**

Bytes and string length are not the same since strings are assumed to be UTF-8 encoded (according to the ABI defintion) therefore one character is not nessesarily encoded in one byte of data.

Example:

```
function length(string memory a) public pure returns(uint) {
    bytes memory x = bytes(a);

    return x.length;
}
```

- **Delete from dynamic array: 'delete' on an array leaves a gap**

Using `delete` on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the length property.

Example:

```
contract arr {
    uint[] array = [1,2,3];

    function removeAtIndex() public returns (uint[] memory) {
        delete array[1];
        return array;
    }
}
```

- **Data Truncated: Division on int/uint values truncates the result**

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Example:

```
function contribute() payable public {
    uint fee = msg.value * uint256(feePercentage / 100);
    fee = msg.value * (p2 / 100);
}
```

11.3 Remix-analyzer

`remix-analyzer` is the library which works underneath of remix-ide `Solidity Static Analysis` plugin.

`remix-analyzer` is an NPM package. It can be used as a library in a solution supporting node.js. Find more information about this type of usage in the remix-analyzer repository

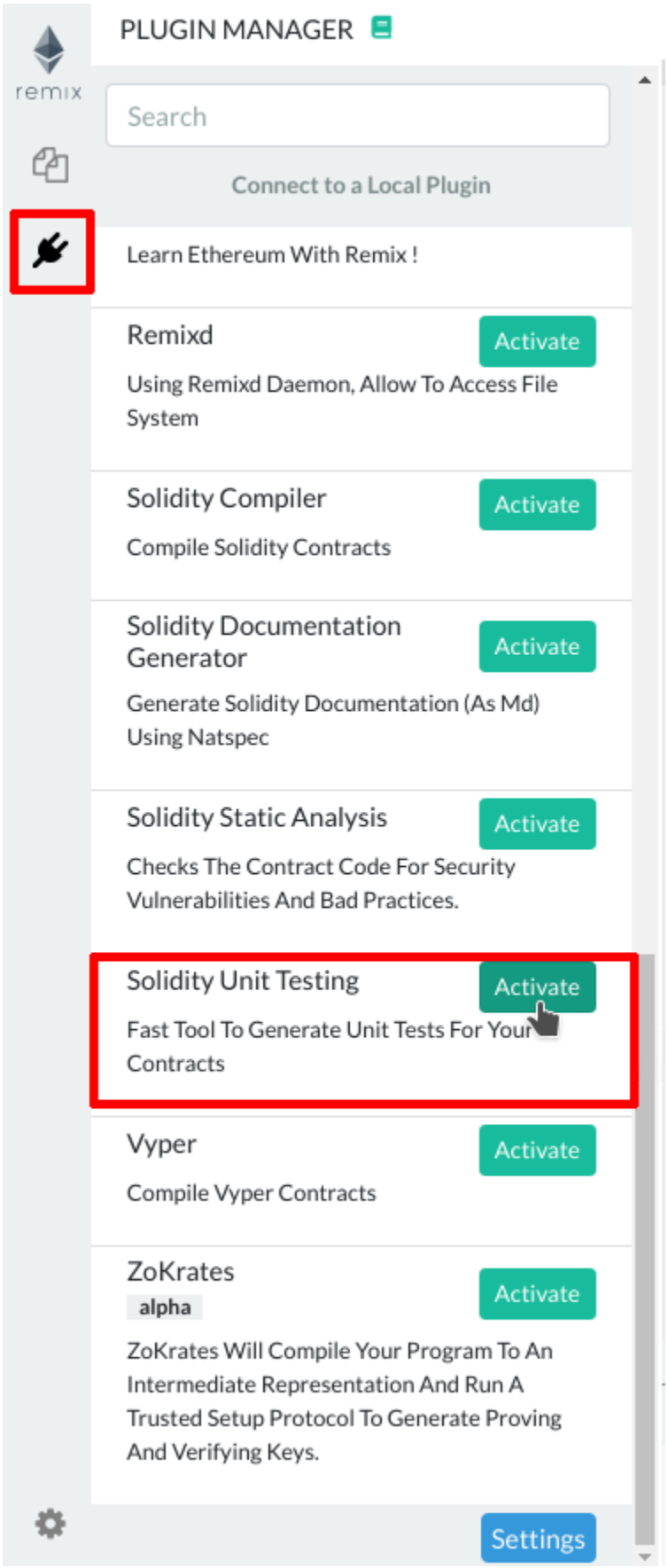
CHAPTER 12

Unit Testing Plugin

Click the  double check icon to get to the `Solidity Unit Testing` plugin.

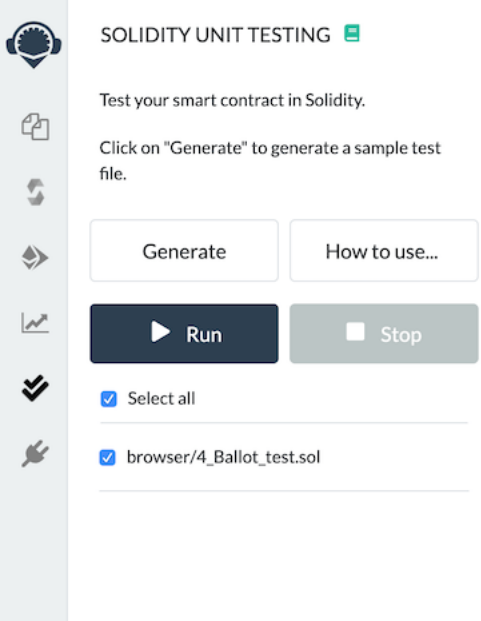
If you haven't used this plugin before and are not seeing `double check` icon, you have to activate it from Remix plugin manager.

Go to the plugin manager (by click the  plug icon) and load up the unit testing plugin.


PLUGIN MANAGER
remix
Search
Connect to a Local Plugin
Learn Ethereum With Remix !
Remixd
Activate
Using Remixd Daemon, Allow To Access File System
Solidity Compiler
Activate
Compile Solidity Contracts
Solidity Documentation Generator
Activate
Generate Solidity Documentation (As Md) Using Natspec
Solidity Static Analysis
Activate
Checks The Contract Code For Security Vulnerabilities And Bad Practices.
Solidity Unit Testing
Activate
Fast Tool To Generate Unit Tests For Your Contracts
Vyper
Activate
Compile Vyper Contracts
ZoKrates
alpha
Activate
ZoKrates Will Compile Your Program To An Intermediate Representation And Run A Trusted Setup Protocol To Generate Proving And Verifying Keys.
Settings

Now `double check` icon will appear on the left side icon bar. Clicking on icon will load the unit testing module in the side panel.

Alternatively, just select `Solidity` environment from remix IDE home page. This will activate `Solidity Unit Testing` plugin along with `Solidity Compiler`, `Deploy & Run Transactions` & `Solidity Static Analysis` plugins.



12.1 Generate

Select a solidity file which you want to test and click on the button `Generate`. It will generate a new sample solidity test file **in the current folder** suffixed with `_test`. This file contains the minimum you need for running unit testing.

12.2 Write Tests

Write tests to check the functionality of your contract. Remix injects a built-in `assert` library which can be used for testing. Visit the library documentation *here*.

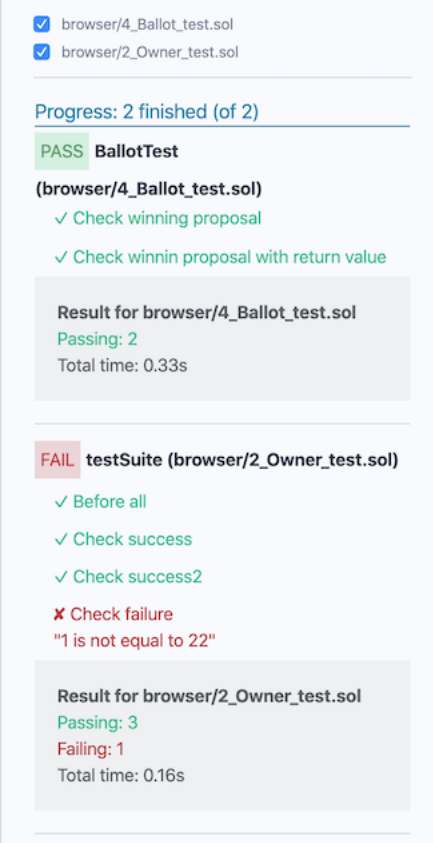
Apart from this, Remix allows usage of some special functions to make testing more structural. They are:

- `beforeEach()` - Runs before each test
- `beforeAll()` - Runs before all tests
- `afterEach()` - Runs after each test
- `afterAll()` - Runs after all tests

To get started, see this simple example.

12.3 Run

Once you are done with writing tests, select the _test.sol files in the list and click on the button `Run` to execute the tests in the selected files. The execution will run in a separate environment and the result will be displayed below.



12.4 Stop

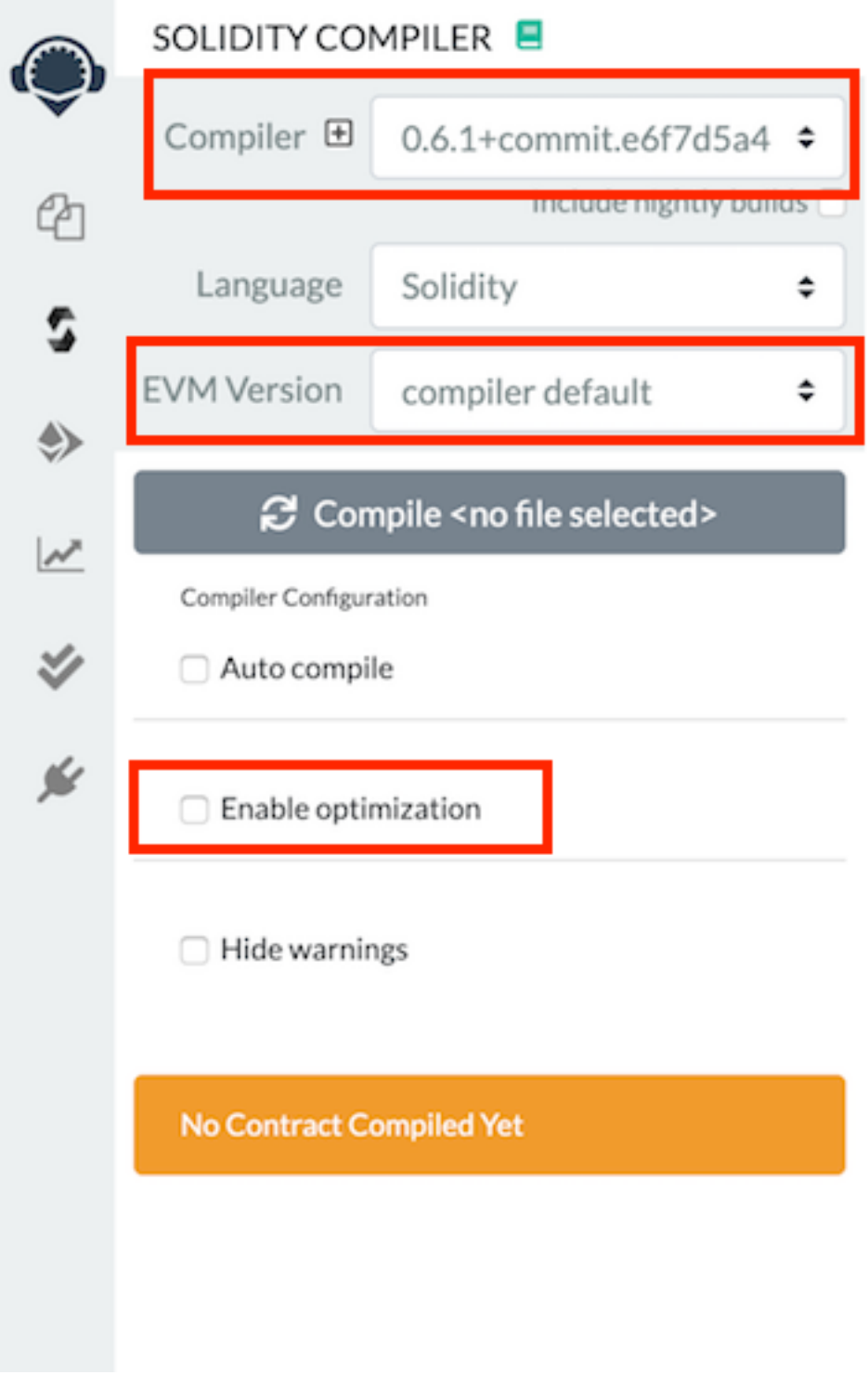
If you have selected multiple files to run the tests and want to stop the execution, click on `Stop` button. It will stop execution after running the tests for current file.

12.5 Customization

Remix facilitates users with various types of customizations to test a contract properly.

1. Custom Compiler Context

`Solidity Unit Testing` refers `Solidity Compiler` plugin for compiler configurations. One can provide customized inputs for `Compiler`, `EVM Version` & `Enable Optimization` and these will be the configuration settings used for contract compilation before running unit tests.



2. Custom Transaction Context

For a contract method interaction, prime parameters of transaction are `from` address, `value` & `gas`. Usually, we need to test a method's behaviour under different values of these parameters.

Remix provides the functionality of custom `msg.sender` & `msg.value` of transaction using method devdoc like:

```
/// #sender: account-0
/// #value: 10
function checkSenderIs0AndValueis10 () public payable {
    Assert.equal(msg.sender, TestsAccounts.getAccount(0), "wrong sender in␣
↪checkSenderIs0AndValueis10");
    Assert.equal(msg.value, 10, "wrong value in checkSenderIs0AndValueis10");
}
```

Things to keep in mind while using custom transaction context:

1. Parameters must be defined in devdoc of related method
2. Each parameter key should be prefixed with a hash (**#**) and end with a colon following a space (**:**) like `#sender:` & `#value:`
3. For now, customization is available for parameters `sender` & `value` only
4. Sender is `from` address of a transaction which is accessed using `msg.sender` inside a contract method. It should be defined in a fixed format as '**account-**<account_index>'
5. `<account_index>` varies from `0-2` before remix-ide release v0.10.0 and `0-9` afterwards
6. `remix_accounts.sol` must be imported in your test file to use custom `sender`
7. Value is `value` sent along with a transaction in `wei` which is accessed using `msg.value` inside a contract method. It should be a number.

Regarding `gas`, Remix estimates the required gas for each transaction internally. Still if a contract deployment fails with `Out-of-Gas` error, it tries to redeploy it by doubling the gas. Deployment failing with double gas will show error: `contract deployment failed after trying twice: The contract code couldn't be stored, please check your gas limit`

Various test examples can be seen in *examples* section.

12.6 Points to remember

- A test contract cannot have a method with parameters. Having one such method will show error: `Method 'methodname' can not have parameters inside a test contract`
- Number of test accounts are `3` before remix-ide release v0.10.0 and `10` afterwards
- A test file which imports `remix_accounts.sol` might not compile successfully with `Solidity Compiler` plugin but it will work fine with Solidity Unit Testing plugin.

12.7 Remix-tests

`remix-tests` is the module which works underneath of remix-ide `Solidity Unit Testing` plugin.

`remix-tests` is an NPM package. It can also be used as a CLI/CI solution, supporting node.js. Find more information about this type of usage in the remix-tests repository

For CI implementation example, see Su Squares contract and Travis build that uses `remix-tests` for continuous integration testing.

CHAPTER 13

Remix Assert Library

13.1 Assert

13.1.1 Assert.ok(value[, message])

- `value`: <bool>
- `message`: <string>

Tests if value is truthy. `message` is returned in case of failure.

Examples:

```
Assert.ok(true);
// OK
Assert.ok(false, "it\'s false");
// error: it's false
```

13.1.2 Assert.equal(actual, expected[, message])

- `actual`: <uint | int | bool | address | bytes32 | string>
- `expected`: <uint | int | bool | address | bytes32 | string>
- `message`: <string>

Tests if `actual` & `expected` values are same. `message` is returned in case of failure.

Examples:

```
Assert.equal(string("a"), "a");
// OK
Assert.equal(uint(100), 100);
// OK
foo.set(200)
Assert.equal(foo.get(), 200);
// OK
Assert.equal(foo.get(), 100, "value should be 200");
// error: value should be 200
```

13.1.3 Assert.notEqual(actual, expected[, message])

- `actual`: <uint | int | bool | address | bytes32 | string>
- `expected`: <uint | int | bool | address | bytes32 | string>
- `message`: <string>

Tests if `actual` & `expected` values are not same. `message` is returned in case of failure.

Examples:

```
Assert.notEqual(string("a"), "b");
// OK
foo.set(200)
Assert.notEqual(foo.get(), 200, "value should not be 200");
// error: value should not be 200
```

13.1.4 Assert.greaterThan(value1, value2[, message])

- `value1`: <uint | int>
- `value2`: <uint | int>
- `message`: <string>

Tests if `value1` is greater than `value2`. `message` is returned in case of failure.

Examples:

```
Assert.greaterThan(uint(2), uint(1));
// OK
Assert.greaterThan(uint(-2), uint(1));
// OK
Assert.greaterThan(int(2), int(1));
// OK
Assert.greaterThan(int(-2), int(-1), "-2 is not greater than -1");
// error: -2 is not greater than -1
```

13.1.5 Assert.lesserThan(value1, value2[, message])

- `value1`: <uint | int>

- `value2`: <uint | int>
- `message`: <string>

Tests if `value1` is lesser than `value2`. `message` is returned in case of failure.

Examples:

```
Assert.lesserThan(int(-2), int(-1));
// OK
Assert.lesserThan(int(2), int(1), "2 is not lesser than 1");
// error: 2 is not greater than 1
```

CHAPTER 14

Testing by Example

Here are some examples which can give you better understanding to plan your tests.

Note: Examples in this section are intended to give you a push for development. We don't recommend to rely on them without verifying at your end.

14.1 1. Simple example

In this example, we test setting & getting variables.

Contract/Program to be tested: `Simple_storage.sol`

```
pragma solidity >=0.4.22 <0.7.0;

contract SimpleStorage {
    uint public storedData;

    constructor() public {
        storedData = 100;
    }

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint retVal) {
        return storedData;
    }
}
```

Test contract/program: `simple_storage_test.sol`

```
pragma solidity >=0.4.22 <0.7.0;
import "remix_tests.sol";
```

(continues on next page)

(continued from previous page)

```
import "./Simple_storage.sol";

contract MyTest {
    SimpleStorage foo;

    // beforeEach works before running each test
    function beforeEach() public {
        foo = new SimpleStorage();
    }

    /// Test if initial value is set correctly
    function initialValueShouldBe100() public returns (bool) {
        return Assert.equal(foo.get(), 100, "initial value is not correct");
    }

    /// Test if value is set as expected
    function valueIsSet200() public returns (bool) {
        foo.set(200);
        return Assert.equal(foo.get(), 200, "value is not 200");
    }
}
```

14.2 2. Testing a method involving `msg.sender`

In Solidity, `msg.sender` plays a great role in access management of a smart contract methods interaction. Different `msg.sender` can help to test a contract involving multiple accounts with different roles. Here is an example for testing such case:

Contract/Program to be tested: `Sender.sol`

```
pragma solidity >=0.4.22 <0.7.0;
contract Sender {
    address private owner;

    constructor() public {
        owner = msg.sender;
    }

    function updateOwner(address newOwner) public {
        require(msg.sender == owner, "only current owner can update owner");
        owner = newOwner;
    }

    function getOwner() public view returns (address) {
        return owner;
    }
}
```

Test contract/program: `Sender_test.sol`

```
pragma solidity >=0.4.22 <0.7.0;
import "remix_tests.sol"; // this import is automatically injected by Remix
import "remix_accounts.sol";
import "./Sender.sol";
```

(continues on next page)

(continued from previous page)

```
// Inherit 'Sender' contract
contract SenderTest is Sender {
    /// Define variables referring to different accounts
    address acc0;
    address acc1;
    address acc2;

    /// Initiate accounts variable
    function beforeAll() public {
        acc0 = TestsAccounts.getAccount(0);
        acc1 = TestsAccounts.getAccount(1);
        acc2 = TestsAccounts.getAccount(2);
    }

    /// Test if initial owner is set correctly
    function testInitialOwner() public {
        // account at zero index (account-0) is default account, so current owner␣
↪should be acc0
        Assert.equal(getOwner(), acc0, 'owner should be acc0');
    }

    /// Update owner first time
    /// This method will be called by default account(account-0) as there is no␣
↪custom sender defined
    function updateOwnerOnce() public {
        // check method caller is as expected
        Assert.ok(msg.sender == acc0, 'caller should be default account i.e. acc0');
        // update owner address to acc1
        updateOwner(acc1);
        // check if owner is set to expected account
        Assert.equal(getOwner(), acc1, 'owner should be updated to acc1');
    }

    /// Update owner again by defining custom sender
    /// #sender: account-1 (sender is account at index '1')
    function updateOwnerOnceAgain() public {
        // check if caller is custom and is as expected
        Assert.ok(msg.sender == acc1, 'caller should be custom account i.e. acc1');
        // update owner address to acc2. This will be successful because acc1 is␣
↪current owner & caller both
        updateOwner(acc2);
        // check if owner is set to expected account i.e. account2
        Assert.equal(getOwner(), acc2, 'owner should be updated to acc2');
    }
}
```

14.3 3. Testing method execution

With Solidity, one can directly verify the changes made by a method in storage by retrieving those variables from a contract. But testing for a successful method execution takes some strategy. Well that is not entirely true, when a test is successful - it is usually obvious why it passed. However, when a test fails, it is essential to understand why it failed.

To help in such cases, Solidity introduced the `try-catch` statement in version `0.6.0`. Previously, we had to use low-level calls to track down what was going on.

Here is an example test file that use both **try-catch** blocks and **low level calls**:

Contract/Program to be tested: `AttendanceRegister.sol`

```
pragma solidity >=0.4.22 <0.7.0;
contract AttendanceRegister {
    struct Student{
            string name;
            uint class;
        }

    event Added(string name, uint class, uint time);

    mapping(uint => Student) public register; // roll number => student details

    function add(uint rollNumber, string memory name, uint class) public returns
↪(uint256){
        require(class > 0 && class <= 12, "Invalid class");
        require(register[rollNumber].class == 0, "Roll number not available");
        Student memory s = Student(name, class);
        register[rollNumber] = s;
        emit Added(name, class, now);
        return rollNumber;
    }

    function getStudentName(uint rollNumber) public view returns (string memory) {
        return register[rollNumber].name;
    }
}
```

Test contract/program: `AttendanceRegister_test.sol`

```
pragma solidity >=0.4.22 <0.7.0;
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "./AttendanceRegister.sol";

contract AttendanceRegisterTest {

    AttendanceRegister ar;

    /// 'beforeAll' runs before all other tests
    function beforeAll () public {
        // Create an instance of contract to be tested
        ar = new AttendanceRegister();
    }

    /// For solidity version greater or equal to 0.6.0,
    /// See: https://solidity.readthedocs.io/en/v0.6.0/control-structures.html#try-
↪catch
    /// Test 'add' using try-catch
    function testAddSuccessUsingTryCatch() public {
        // This will pass
        try ar.add(101, 'secondStudent', 11) returns (uint256 r) {
            Assert.equal(r, 101, 'wrong rollNumber');
        } catch Error(string memory /*reason*/) {
            // This is executed in case
            // revert was called inside getData
            // and a reason string was provided.
```

(continues on next page)

(continued from previous page)

```
            Assert.ok(false, 'failed with reason');
        } catch (bytes memory /*lowLevelData*/) {
            // This is executed in case revert() was used
            // or there was a failing assertion, division
            // by zero, etc. inside getData.
            Assert.ok(false, 'failed unexpected');
        }
    }

    /// Test failure case of 'add' using try-catch
    function testAddFailureUsingTryCatch1() public {
        // This will revert on 'require(class > 0 && class <= 12, "Invalid class");'␣
↪for class '13'
        try ar.add(101, 'secondStudent', 13) returns (uint256 r) {
            Assert.ok(false, 'method execution should fail');
        } catch Error(string memory reason) {
            // Compare failure reason, check if it is as expected
            Assert.equal(reason, 'Invalid class', 'failed with unexpected reason');
        } catch (bytes memory /*lowLevelData*/) {
            Assert.ok(false, 'failed unexpected');
        }
    }

    /// Test another failure case of 'add' using try-catch
    function testAddFailureUsingTryCatch2() public {
        // This will revert on 'require(register[rollNumber].class == 0, "Roll number␣
↪not available");' for rollNumber '101'
        try ar.add(101, 'secondStudent', 11) returns (uint256 r) {
            Assert.ok(false, 'method execution should fail');
        } catch Error(string memory reason) {
            // Compare failure reason, check if it is as expected
            Assert.equal(reason, 'Roll number not available', 'failed with unexpected␣
↪reason');
        } catch (bytes memory /*lowLevelData*/) {
            Assert.ok(false, 'failed unexpected');
        }
    }

    /// For solidity version less than 0.6.0, low level call can be used
    /// See: https://solidity.readthedocs.io/en/v0.6.0/units-and-global-variables.html
↪#members-of-address-types
    /// Test success case of 'add' using low level call
    function testAddSuccessUsingCall() public {
        bytes memory methodSign = abi.encodeWithSignature('add(uint256,string,uint256)
↪', 102, 'firstStudent', 10);
        (bool success, bytes memory data) = address(ar).call(methodSign);
        // 'success' stores the result in bool, this can be used to check whether␣
↪method call was successful
        Assert.equal(success, true, 'execution should be successful');
        // 'data' stores the returned data which can be decoded to get the actual␣
↪result
        uint rollNumber = abi.decode(data, (uint256));
        // check if result is as expected
        Assert.equal(rollNumber, 102, 'wrong rollNumber');
    }

    /// Test failure case of 'add' using low level call
```

(continues on next page)

(continued from previous page)

```
    function testAddFailureUsingCall() public {
        bytes memory methodSign = abi.encodeWithSignature('add(uint256,string,uint256)
↪', 102, 'duplicate', 10);
        (bool success, bytes memory data) = address(ar).call(methodSign);
        // 'success' will be false if method execution is not successful
        Assert.equal(success, false, 'execution should be successful');
    }
}
```

14.4 4. Testing a method involving `msg.value`

In Solidity, ether can be passed along with a method call which is accessed inside contract as `msg.value`. Sometimes, multiple calculations in a method are performed based on `msg.value` which can be tested with various values using Remix's Custom transaction context. See the example:

Contract/Program to be tested: `Value.sol`

```
pragma solidity >=0.4.22 <0.7.0;
contract Value {
    uint256 public tokenBalance;

    constructor() public {
        tokenBalance = 0;
    }

    function addValue() payable public {
        tokenBalance = tokenBalance + (msg.value/10);
    }

    function getTokenBalance() view public returns (uint256) {
        return tokenBalance;
    }
}
```

Test contract/program: `Value_test.sol`

```
pragma solidity >=0.4.22 <0.7.0;
import "remix_tests.sol";
import "./Value.sol";

contract ValueTest{
    Value v;

    function beforeAll() public {
        // create a new instance of Value contract
        v = new Value();
    }

    /// Test initial balance
    function testInitialBalance() public {
        // initially token balance should be 0
        Assert.equal(v.getTokenBalance(), 0, 'token balance should be 0 initially');
    }
```

(continues on next page)

(continued from previous page)

```
    /// For Solidity version greater than 0.6.1
    /// Test 'addValue' execution by passing custom ether amount
    /// #value: 200
    function addValueOnce() public payable {
        // check if value is same as provided through devdoc
        Assert.equal(msg.value, 200, 'value should be 200');
        // execute 'addValue'
        v.addValue{gas: 40000, value: 200}(); // introduced in Solidity version 0.6.2
        // As per the calculation, check the total balance
        Assert.equal(v.getTokenBalance(), 20, 'token balance should be 20');
    }

    /// For Solidity version less than 0.6.2
    /// Test 'addValue' execution by passing custom ether amount again using low␣
↪level call
    /// #value: 100
    function addValueAgain() public payable {
        Assert.equal(msg.value, 100, 'value should be 100');
        bytes memory methodSign = abi.encodeWithSignature('addValue()');
        (bool success, bytes memory data) = address(v).call.gas(40000).
↪value(100)(methodSign);
        Assert.equal(success, true, 'execution should be successful');
        Assert.equal(v.getTokenBalance(), 30, 'token balance should be 30');
    }
}
```

CHAPTER 15

Build Artifact

When a compilation succeeds, Remix creates two JSON files for each compiled contract. One of these files **captures the output from the Solidity compilation**. This file will be named **contractName_metadata.json**.

The other JSON file is named **contractName.json** . The **contractName.json** file contains the compilation's artifact that is needed for linking a library to the file. It contains the link to the libraries, the bytecode, the deployed bytecode, the gas estimation, the method identifiers, and the ABI.

In order to generate these artifact files, the **Generate contract metadata** box in the **General settings** section of the **Settings** module needs to be checked. The these metadatas files will then be generated when you compile a file and will be placed in the **artifacts** folder - which you can see in the Files Explorers plugin.

You can write scripts that can access either of these files.

15.1 Library Deployment with filename.json

By default Remix automatically deploys needed libraries.

When you open the metadata file for the libraries - **artifact/filename.json** you will see the following sections:

`linkReferences` contains a map representing libraries which depend on the current contract. Values are addresses of libraries used for linking the contract.

`autoDeployLib` defines if the libraries should be auto deployed by Remix or if the contract should be linked with libraries described in `linkReferences`

Note that Remix will resolve addresses corresponding to the current network. By default, a configuration key follows the form: `<network_name>:<networkd_id>`, but it is also possible to define `<network_name>` or `<network_id>` as keys.

Here is a sample metadata file for linking a library:

```
{
        "VM:-": {
                "linkReferences": {
```

(continues on next page)

(continued from previous page)


```
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "main:1": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "ropsten:3": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "rinkeby:4": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "kovan:42": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "data": {
                "bytecode": {
                        "linkReferences": {},
                        "object":
↪"6080604052348015610010576000080fd5b506040516108723803806108728339818101604052602081101561003357600
↪",
                        "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1␣
↪ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x40 MLOAD PUSH2␣
↪0x872 CODESIZE SUB DUP1 PUSH2 0x872 DUP4 CODECOPY DUP2 DUP2 ADD PUSH1 0x40 MSTORE␣
↪PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x33 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD␣
↪SWAP1 DUP1 DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP␣
↪CALLER PUSH1 0x0 DUP1 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND SWAP1 DUP4 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND MUL OR SWAP1 SSTORE POP PUSH1 0x1␣
↪DUP1 PUSH1 0x0 DUP1 PUSH1 0x0 SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1␣
↪DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 ADD DUP2 SWAP1 SSTORE POP␣
↪DUP1 PUSH1 0xFF AND PUSH1 0x2 DUP2 PUSH2 0xFD SWAP2 SWAP1 PUSH2 0x104 JUMP JUMPDEST␣
↪POP POP PUSH2 0x157 JUMP JUMPDEST DUP2 SLOAD DUP2 DUP4 SSTORE DUP2 DUP2 GT ISZERO␣
```


(continues on next page)

(continued from previous page)


```
                        "sourceMap": "33:2130:0:-;;;382:163;8:9:-1;5:2;;;30:1;27;
↪20:12;5:2;382:163:0;;;;;;;;;;;;;;;13:2:-1;8:3;5:11;2:2;;;29:1;26;19:12;2:2;
↪382:163:0;;;;;;;;;;;;;;;446:10;432:11;;:24;;;;;;;;;;;;;;;;;;495:1;466:6;:19;473:11;
↪;;;;;;;;;;466:19;;;;;;;;;;;;;;;:26;;:30;;;;525:13;506:32;;:9;:32;;;;;;;:i;;;:-;;
↪382:163;33:2130;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;:i;;;:-;;;;;;;:o;;;:-;;;;;;;;;;;;;;;;
↪;;;;;;;;;;;;;;;:o;;;:-;;;;;;;"
                },
                "deployedBytecode": {
                        "linkReferences": {},
                        "object":
↪"608060405234801561001057600080fd5b506004361061004c5760003560e01c80635c19a95c1461005157806309ff1bc
↪",
                        "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1␣
↪ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE␣
↪LT PUSH2 0x4C JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x5C19A95C EQ␣
↪PUSH2 0x51 JUMPI DUP1 PUSH4 0x609FF1BD EQ PUSH2 0x95 JUMPI DUP1 PUSH4 0x9E7B8D61 EQ␣
↪PUSH2 0xB9 JUMPI DUP1 PUSH4 0xB3F98ADC EQ PUSH2 0xFD JUMPI JUMPDEST PUSH1 0x0 DUP1␣
↪REVERT JUMPDEST PUSH2 0x93 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT␣
↪ISZERO PUSH2 0x67 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1␣
↪CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20␣
↪ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x12E JUMP JUMPDEST STOP JUMPDEST␣
↪PUSH2 0x9D PUSH2 0x481 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP3 PUSH1 0xFF AND␣
↪PUSH1 0xFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP POP PUSH1 0x40 MLOAD DUP1 SWAP2␣
↪SUB SWAP1 RETURN JUMPDEST PUSH2 0xFB PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20␣
↪DUP2 LT ISZERO PUSH2 0xCF JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1␣
↪DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1␣
↪0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x4F9 JUMP JUMPDEST STOP␣
↪JUMPDEST PUSH2 0x12C PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO␣
↪PUSH2 0x113 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1␣
↪CALLDATALOAD PUSH1 0xFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP␣
↪POP PUSH2 0x5F6 JUMP JUMPDEST STOP JUMPDEST PUSH1 0x0 PUSH1 0x1 PUSH1 0x0 CALLER␣
↪PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1␣
↪DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SWAP1 POP DUP1 PUSH1 0x1 ADD PUSH1␣
↪0x0 SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH1 0xFF AND ISZERO PUSH2 0x18E␣
↪JUMPI POP PUSH2 0x47E JUMP JUMPDEST JUMPDEST PUSH1 0x0 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1␣
↪DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x1 ADD PUSH1 0x2 SWAP1 SLOAD␣
↪SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF␣
↪AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO DUP1 ISZERO␣
↪PUSH2 0x2BC JUMPI POP CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND␣
↪PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND␣
↪PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD␣
↪SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x1 ADD PUSH1 0x2 SWAP1␣
↪SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO JUMPDEST ISZERO PUSH2␣
↪0x32B JUMPI PUSH1 0x1 PUSH1 0x0 DUP4 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1␣
↪DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x1 ADD PUSH1 0x2 SWAP1 SLOAD␣
↪SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF␣
↪AND SWAP2 POP PUSH2 0x18F JUMP JUMPDEST CALLER PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP3 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO PUSH2 0x365 JUMPI POP␣
↪PUSH2 0x47E JUMP JUMPDEST PUSH1 0x1 DUP2 PUSH1 0x1 ADD PUSH1 0x0 PUSH2 0x100 EXP␣
↪DUP2 SLOAD DUP2 PUSH1 0xFF MUL NOT AND SWAP1 DUP4 ISZERO ISZERO MUL OR SWAP1 SSTORE␣
↪POP DUP2 DUP2 PUSH1 0x1 ADD PUSH1 0x2 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND SWAP1 DUP4 PUSH20␣
↪0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND MUL OR SWAP1 SSTORE POP PUSH1 0x0␣
↪PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND␣
↪PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD␣
```


(continues on next page)

(continued from previous page)


```
                        "sourceMap": "33:2130:0:-;;;;8:9:-1;5:2;;;30:1;27;20:12;5:2;
↪33:2130:0;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;872:577;;;;;;13:2:-1;8:3;5:11;2:2;;;29:1;
↪26;19:12;2:2;872:577:0;;;;;;;;;;;;;;;;;;;;;;:::i;:::-;;1801:360;;;;;;i;:::-;;;;;;;;;;;;
↪;;;;;;;;;;;655:164;;;;;;13:2:-1;8:3;5:11;2:2;;;29:1;26;19:12;2:2;655:164:0;;;;;;;;;;
↪;;;;;;;;;;;:::i;:::-;;1509:286;;;;;;13:2:-1;8:3;5:11;2:2;;;29:1;26;19:12;2:2;
↪1509:286:0;;;;;;;;;;;;;;;;;;;;;;:::i;:::-;;872:577;919:20;942:6;:18;949:10;942:18;;;;;;
↪;;;;;;;;;919:41;;995:6;:12;;;;;;;;;;;;991:25;;;1009:7;;;991:25;1025:115;1063:1;
↪1032:33;;;6;:10;1039:2;1032:10;;;;;;;;;;;;;;;:19;;;;;;;;;;;;:33;;;;:70;;;;;1092:10;
↪1069:33;;;6;:10;1076:2;1069:10;;;;;;;;;;;;;;;:19;;;;;;;;;;;;:33;;;;1032:70;1025:115;
↪;;1121:6;:10;1128:2;1121:10;;;;;;;;;;;;;;;:19;;;;;;;;;;;;1116:24;;1025:115;;;
↪1160:10;1154:16;;;2;:16;;;1150:29;;;1172:7;;;1150:29;1203:4;1188:6;:12;;;;:19;;;;;;;;
↪;;;;;;;;;;1235:2;1217:6;:15;;;:20;;;;;;;;;;;;;;;;;;1247:24;1274:6;:10;1281:2;
↪1274:10;;;;;;;;;;;;;;;;1247:37;;1298:10;:16;;;;;;;;;;;;1294:148;;;1368:6;:13;;;
↪1328:9;1338:10;:15;;;;;;;;;;;;1328:26;;;;;;;;;;;;;;;;;:36;;;;:53;;;;;;;;;;;1294:148;;
↪;1429:6;:13;;;1408:10;:17;;;;:34;;;;;;;;;;;1294:148;872:577;;;;;;;:o;1801:360::-;
↪1849:22;1883:24;1910:1;1883:28;;1926:10;1939:1;1926:14;;1921:234;1949:9;:16;;;;
↪1942:4;:23;;;1921:234;;;2019:16;1991:9;2001:4;1991:15;;;;;;;;;;;;;;;;;:25;;;:44;
↪1987:168;;;2074:9;2084:4;2074:15;;;;;;;;;;;;;;;;;:25;;;2055:44;;2136:4;2117:23;;
↪1987:168;1967:6;;;;;;;1921:234;;;;1801:360;;;;:o;655:164::-;732:11;;;;;;;;;;;718:25;
↪;:10;:25;;;;:50;;;;747:6;:15;754:7;747:15;;;;;;;;;;;;;;;:21;;;;;;;;;;;;718:50;
↪714:63;;;770:7;;714:63;811:1;786:6;:15;793:7;786:15;;;;;;;;;;;;;;;:22;;:26;;;;
↪655:164;;;;:o;1509:286::-;1558:20;1581:6;:18;1588:10;1581:18;;;;;;;;;;;;;;;1558:41;;
↪1613:6;:12;;;;;;;;;;;;;:46;;;;1643:9;:16;;;;1629:10;:30;;;;1613:46;1609:59;;;1661:7;;
↪;1609:59;1692:4;1677:6;:12;;;;:19;;;;;;;;;;;;;;;;;;1720:10;1706:6;:11;;;;:24;;;;;;;;;
↪;;;;;;;;1775:6;:13;;;1740:9;1750:10;1740:21;;;;;;;;;;;;;;;;;;;:31;;;;:48;;;;;;;;;;;
↪1509:286;;;;;:o"
                },
                "gasEstimates": {
                        "creation": {
                                "codeDepositCost": "360800",
                                "executionCost": "infinite",
                                "totalCost": "infinite"
                        },
                        "external": {
                                "delegate(address)": "infinite",
                                "giveRightToVote(address)": "20997",
                                "vote(uint8)": "62215",
                                "winningProposal()": "infinite"
                        }
                },
                "methodIdentifiers": {
                        "delegate(address)": "5c19a95c",
                        "giveRightToVote(address)": "9e7b8d61",
                        "vote(uint8)": "b3f98adc",
                        "winningProposal()": "609ff1bd"
                }
        },
        "abi": [
                {
                        "constant": false,
                        "inputs": [
                                {
                                        "internalType": "address",
                                        "name": "to",
                                        "type": "address"
                                }
                        ],
```


(continues on next page)

(continued from previous page)

```
                "name": "delegate",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "constant": true,
                "inputs": [],
                "name": "winningProposal",
                "outputs": [
                        {
                                "internalType": "uint8",
                                "name": "_winningProposal",
                                "type": "uint8"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {
                "constant": false,
                "inputs": [
                        {
                                "internalType": "address",
                                "name": "toVoter",
                                "type": "address"
                        }
                ],
                "name": "giveRightToVote",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "constant": false,
                "inputs": [
                        {
                                "internalType": "uint8",
                                "name": "toProposal",
                                "type": "uint8"
                        }
                ],
                "name": "vote",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "inputs": [
                        {
                                "internalType": "uint8",
                                "name": "_numProposals",
                                "type": "uint8"
```

(continues on next page)

(continued from previous page)

```
                                    }
                            ],
                            "payable": false,
                            "stateMutability": "nonpayable",
                            "type": "constructor"
                    }
            ]

}
```

CHAPTER 16

Creating and Deploying a Contract

There are 3 type of environments Remix can be plugged to: `Javascript VM`, `Injected provider`, or `Web3 provider`. (for details see Running transactions)

Both `Web3 provider` and `Injected provider` require the use of an external tool.

The external tool for `Web3 provider` is an Ethereum node and for `Injected provider` Metamask.

The `JavaScript VM` mode is convenient because each execution runs in your browser and you don't need any other software or Ethereum node to run it.

So, it is the easiest test environment - **no setup required!**

But keep in mind that reloading the browser when you are in the Javascript VM will restart Remix in an empty state.

For performance purposes (which is to say - for testing in an environment that is closest to the mainnet), it might also be better to use an external node.

16.1 Selecting the VM mode

Make sure the VM mode is selected. All accounts displayed in `Accounts` should have 100 ether.

16.2 Sample contract

```
pragma solidity ^0.5.1;

contract testContract {

    uint value;

    constructor (uint _p) public {
        value = _p;
    }
```

(continues on next page)

(continued from previous page)

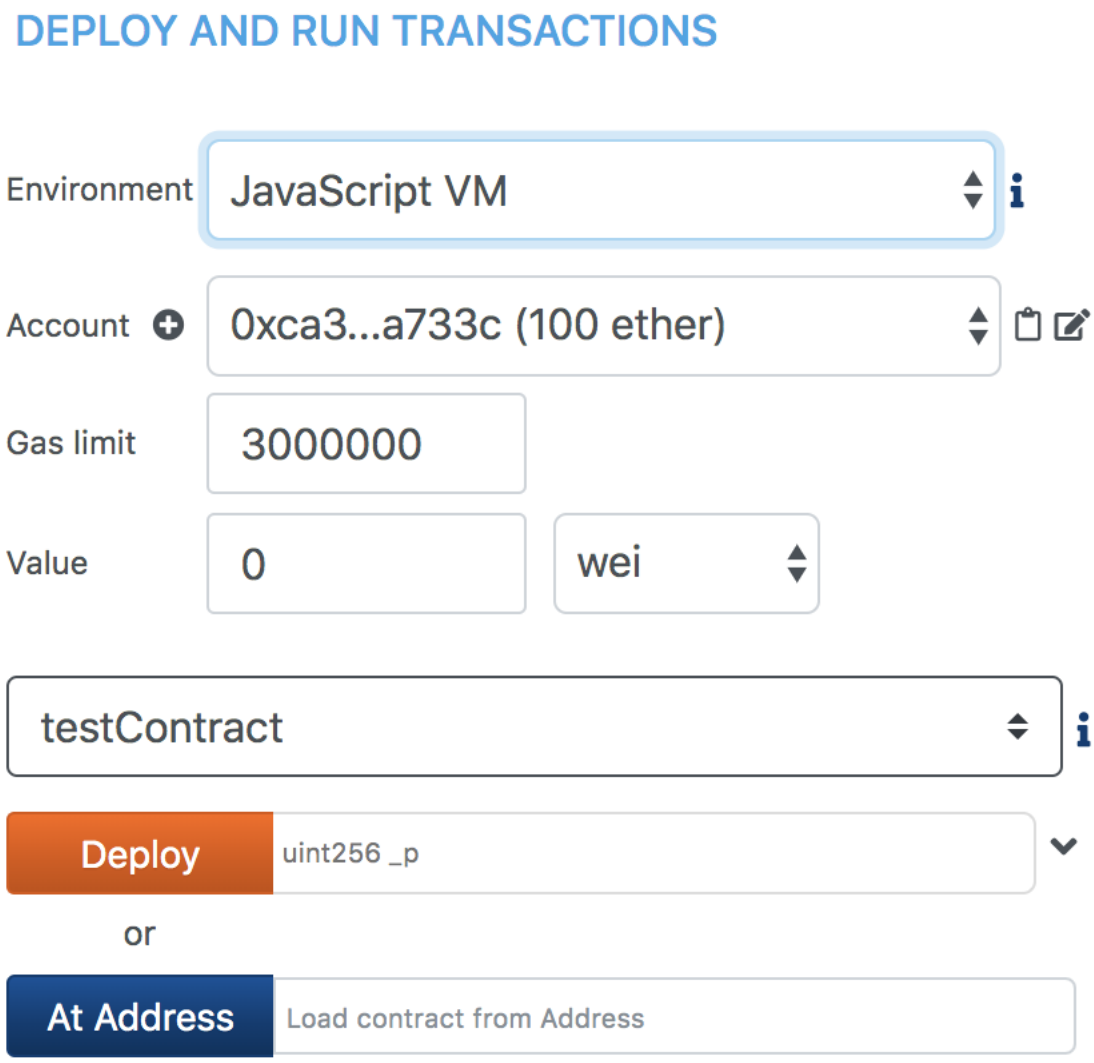
```
    function setP(uint _n) payable public {
        value = _n;
    }

    function setNP(uint _n) public {
        value = _n;
    }

    function get () view public returns (uint) {
        return value;
    }
}
```

This contract is very basic. The goal is to quickly start to create and to interact with a sample contract.

16.3 Deploying an instance

The `Compile tab` displays information related to the current contract (note that there can be more than one) (see compile).

Moving on, in the `Run tab` select, `JavaScript VM` to specify that you are going to deploy an instance of the contract in the `JavaScript VM` state.



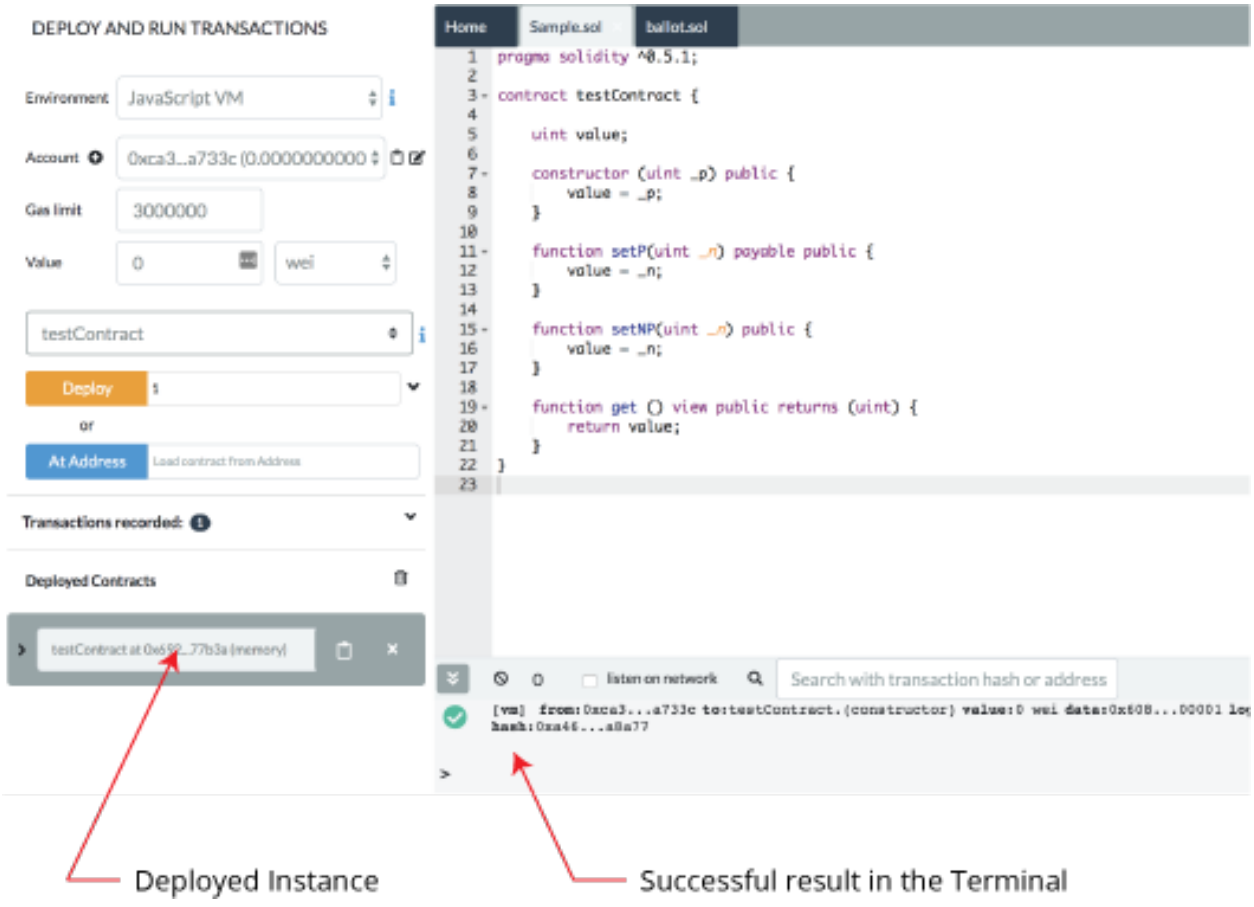
The constructor of `Ballot.sol` needs a parameter (of type `uint8`). Give any value and click on `Deploy`.

The transaction which deploys the instance of `Ballot` is created.

In a "normal" blockchain, it can take several seconds to execute. This is the time for the transaction to be mined. However, because we are using the `JavaScript VM`, our execution is immediate.

The terminal will inform you about the transaction. You can see details there and start debugging.

The newly created instance is displayed in the `run tab`.



16.4 Interacting with an instance

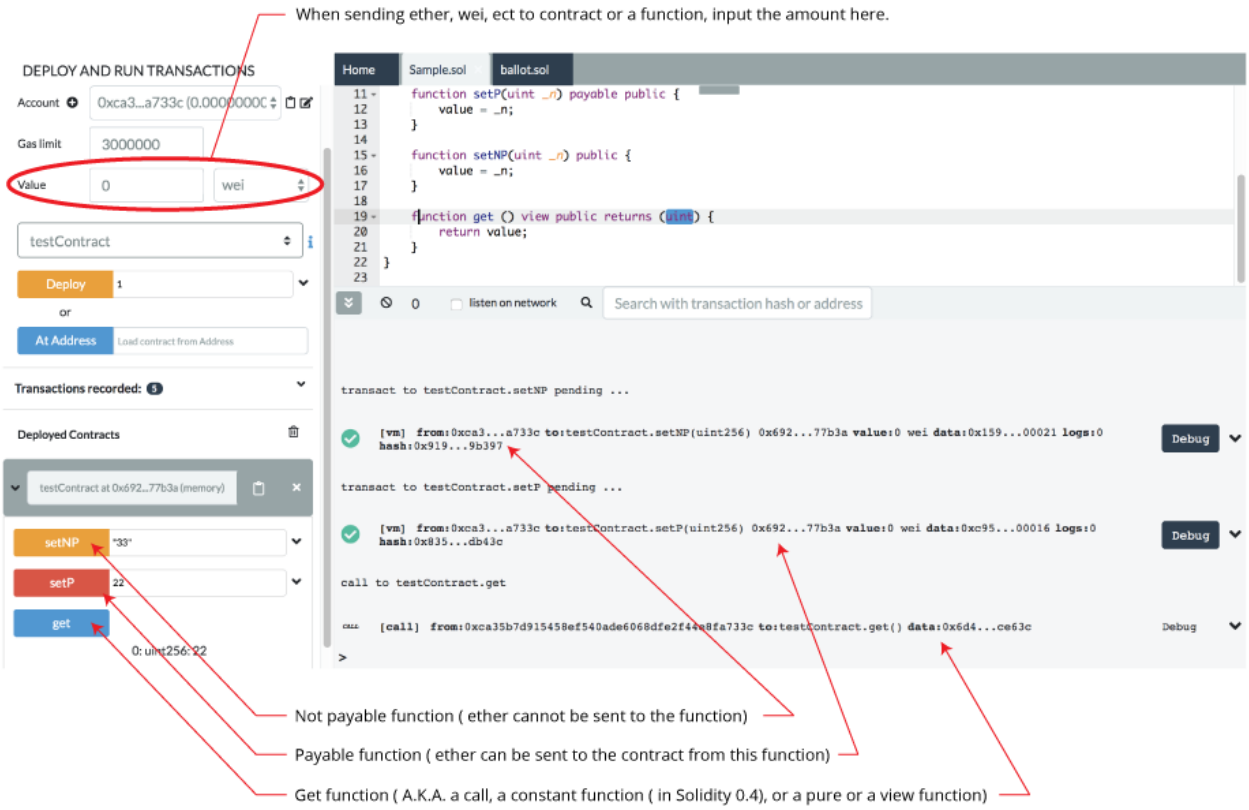
This new instance contains 3 actions which corresponds to the 3 functions (`setP`, `setPN`, `get`). Clicking on `SetP` or `SetPN` will create a new transaction.

Note that `SetP` is `payable` (red button) : it is possible to send value (Ether) to the contract.

`SetPN` is not payable (orange button - depending on the theme) : it is not possible to send value (Ether) to the contract.

Clicking on `get` will not execute a transaction (usually its a blue button - depending on the theme). It doesn't execute a transaction because a `get` does not modify the state (variable `value`) of this instance.

As `get` is `view` you can see the return value just below the action.


When sending ether, wei, ect to contract or a function, input the amount here.
DEPLOY AND RUN TRANSACTIONS
Account
Gas limit
3000000
Value
0
wei
testContract
Deploy
1
or
At Address
Load contract from Address
Transactions recorded:
Deployed Contracts
testContract at 0x692...77b3a (memory)
setNP
"33"
setP
22
get
0: uint256: 22
Home
Sample.sol
ballot.sol
function setP(uint _n) payable public {
value = _n;
}
function setNP(uint _n) public {
value = _n;
}
function get () view public returns (uint) {
return value;
}
}
listen on network
Search with transaction hash or address
transact to testContract.setNP pending ...
[vm] from:0xca3...a733c to:testContract.setNP(uint256) 0x692...77b3a value:0 wei data:0x159...00021 logs:0 hash:0x919...9b397
Debug
transact to testContract.setP pending ...
[vm] from:0xca3...a733c to:testContract.setP(uint256) 0x692...77b3a value:0 wei data:0xc95...00016 logs:0 hash:0x835...db43c
Debug
call to testContract.get
[call] from:0xca35b7d915458ef540ade6068dfe2f44e8fa733c to:testContract.get() data:0x6d4...ce63c
Debug
Not payable function (ether cannot be sent to the function)
Payable function (ether can be sent to the contract from this function)
Get function (A.K.A. a call, a constant function (in Solidity 0.4), or a pure or a view function)

CHAPTER 17

Debugging Transactions

There are two ways to start debugging, each one corresponds to a different use case.

- from the transaction log in the Terminal - use this when you want to debug a transaction.
- from the Debugger - use this if you have a *transaction hash*.

17.1 Initiate Debugging from the transaction log in the Terminal

Let's start with a basic contract (or replace this one by your own) :

- create a blank file in the file explorer (by clicking the + icon) and give it a name.
- copy the code below.
- compile the code.
- click the Run & Deploy icon in the icon panel.

```
pragma solidity >=0.5.1 <0.6.0;
contract Donation {
    address owner;
    event fundMoved(address _to, uint _amount);
    modifier onlyowner { if (msg.sender == owner) _; }
    address[] _giver;
    uint[] _values;

    constructor() public {
        owner = msg.sender;
    }

    function donate() payable  public {
        addGiver(msg.value);
    }
```

(continues on next page)

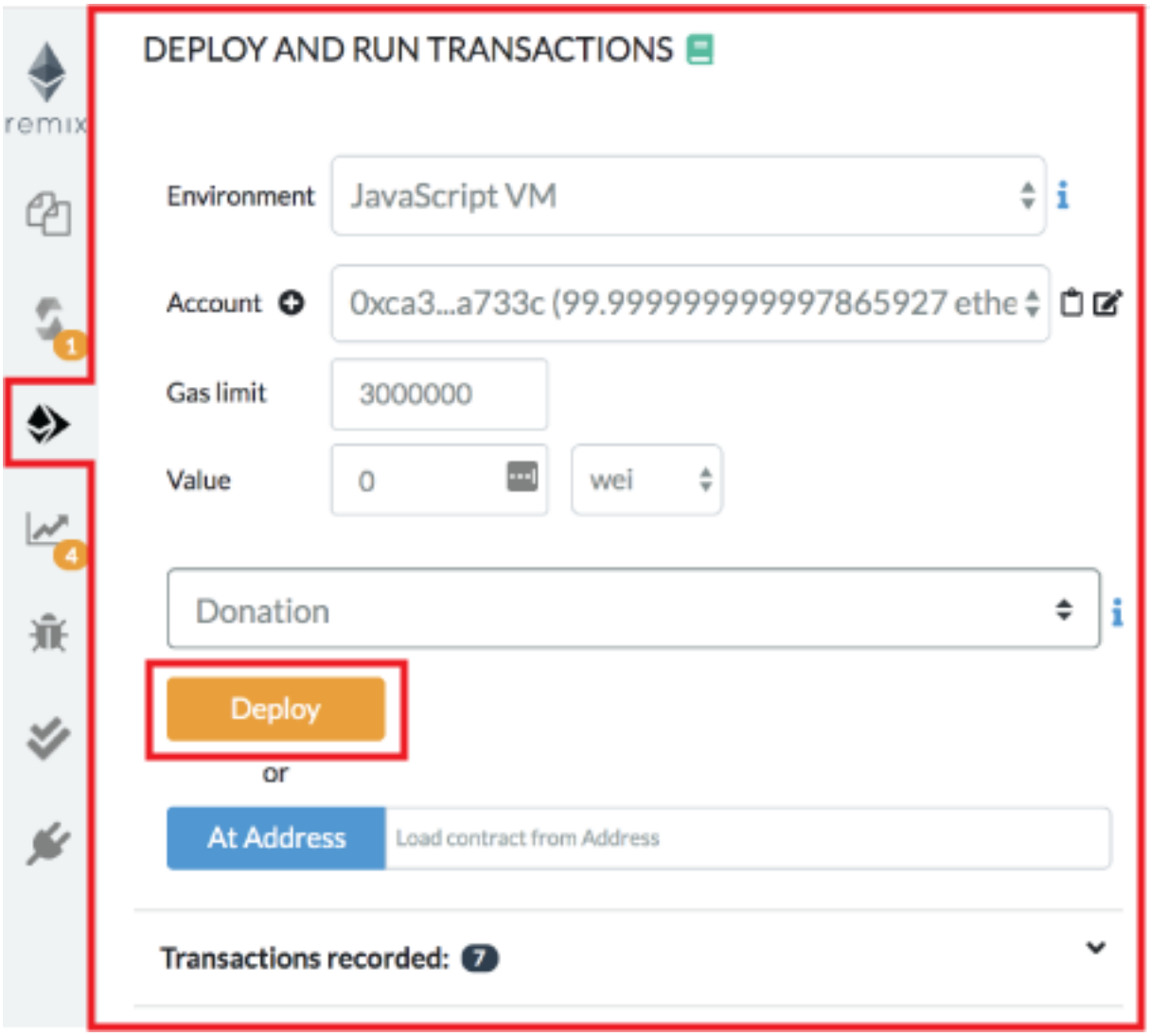
(continued from previous page)

```
    function moveFund(address payable _to, uint _amount) onlyowner  public {
        uint balance = address(this).balance;
        uint amount = _amount;
        if (_amount <= balance) {
            if (_to.send(balance)) {
                emit fundMoved(_to, _amount);
            } else {
               revert();
            }
        } else {
            revert();
        }
    }

    function addGiver(uint _amount) internal {
        _giver.push(msg.sender);
        _values.push(_amount);
    }
}
```

For the purpose of this tutorial, we will run the `JavaScript VM`. This simulates a custom blockchain. You could do the same using a proper backend node.

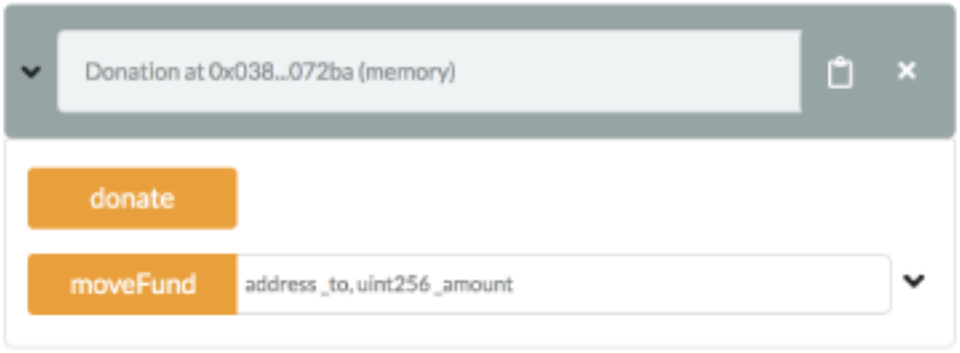
Let's deploy the contract:

Click the `Deploy` button



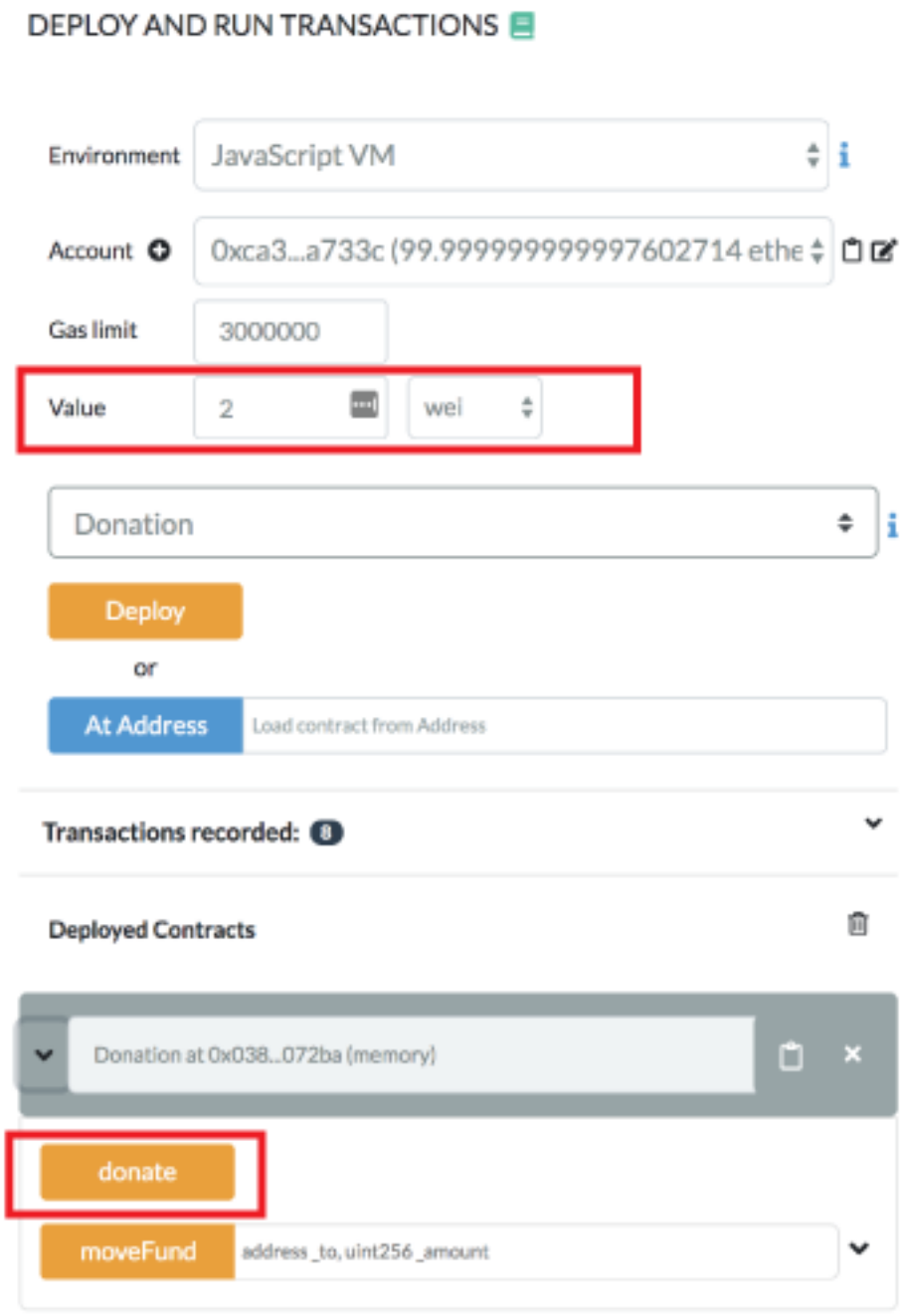
You'll see the deployed instance (AKA the udapp).

Then open it up (by clicking the caret).



We are going to call the `Donate` function and will send it ether.

To do this: in the value input box put in **2** and select Ether as the unit (and not wei like I did in the image below - well you could - it won't really change anything).



Then click the `Donate` button.

This will send Ether to the this function.

Because we are using the `JavaScript VM`, everything happens almost instantly. (If we had been using Injected Web 3, then we would have to need to approve the transaction, pay for gas and wait for the transaction to get mined.)

Remix displays information related to each transaction result in the terminal.

Check in the **terminal** where the transaction you just made is logged.

Click the debug button to start debugging it.

Before we get to the actual debugging tool, the next section show how to start debugging session directly from the Debugger.

17.2 Initiate Debugging from the Debugger

Click the bug icon in the icon panel to get to the debugger in the side panel.

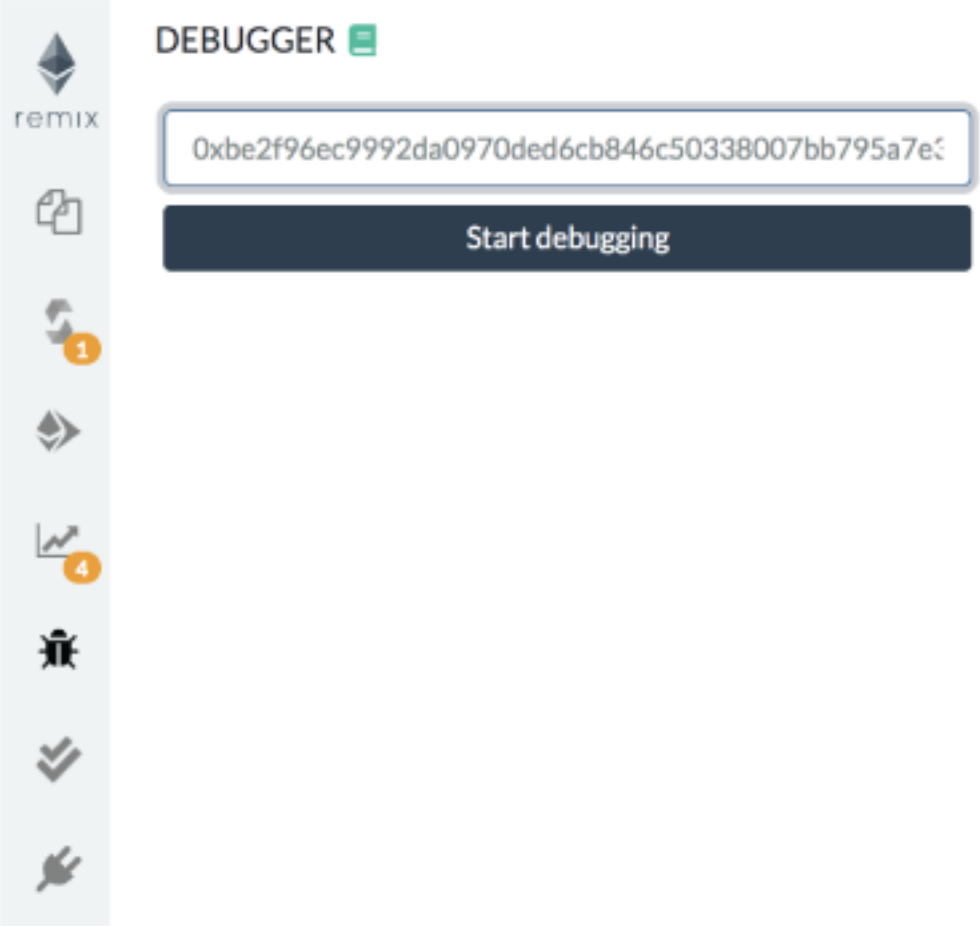
If you don't see the bug icon, go to the plugin manager and activate the debugger.

You can start a debug session by providing a `transaction hash`.

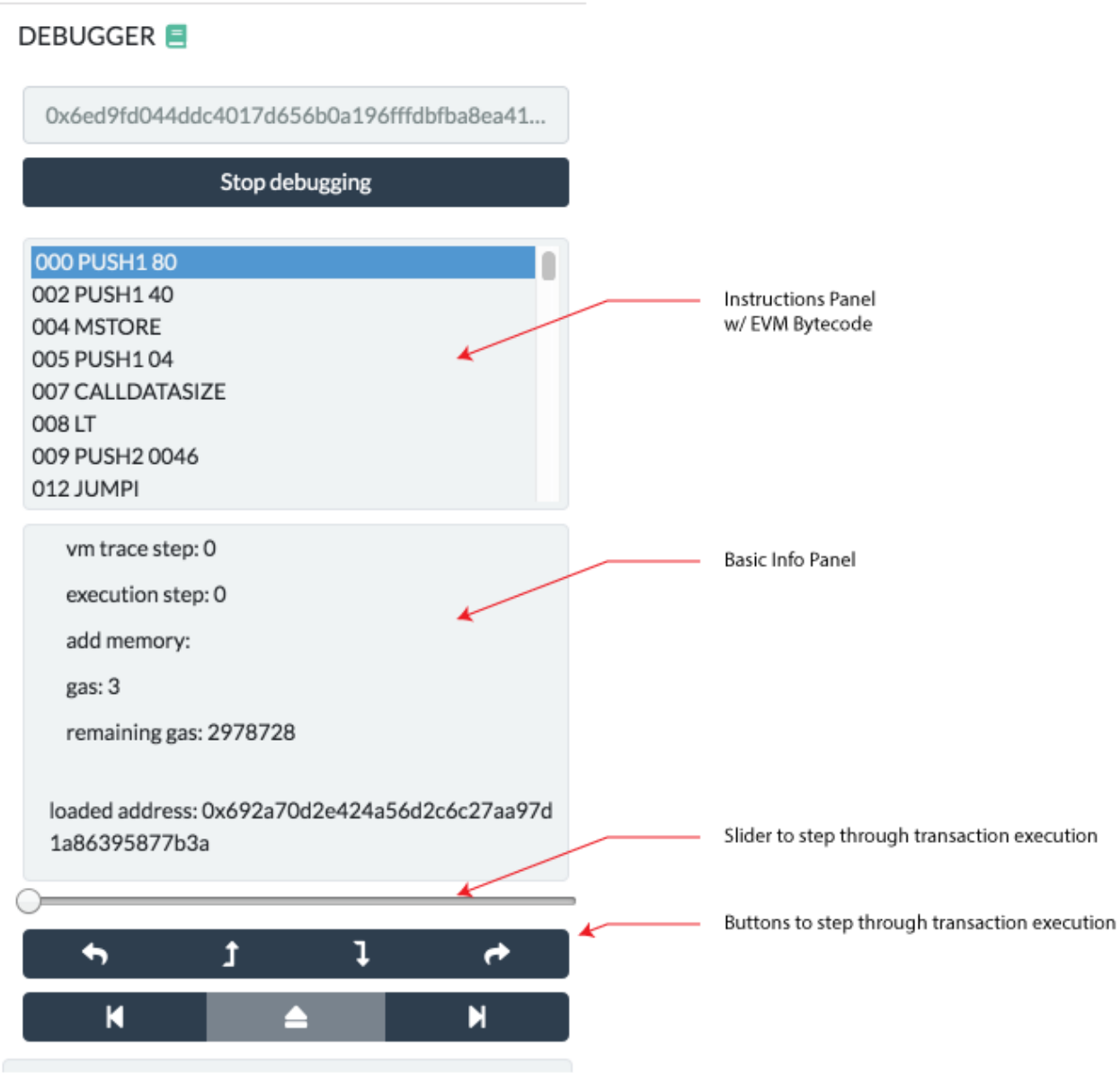
To find a transaction hash:

1. Go to a transaction in the terminal.
2. Click a line with a transaction - to exand the log.
3. The transaction hash is there - copy it.

Then click in the debugger paste the hash and click on the `Start debugging` button.


remix
DEBUGGER
0xbe2f96ec9992da0970ded6cb846c50338007bb795a7e
Start debugging
1
4

17.3 Using the debugger



The debugger allows one to see detailed informations about the transaction's execution. It uses the editor to display the location in the source code where the current execution is.

The navigation part contains a slider and buttons that can be used to step through the transaction execution.

17.3.1 More explaination of what these buttons do.

1. Step Into
2. Step Over Into

11 panels give detailed information about the execution:

17.3.2 Instructions

The Instructions panel displays the bytecode of the current executing contract- with the current step highlighted.

Important note: When this panel is hidden, the slider will have a courser granularity and only stop at *expression boundaries*, even if they are compiled into multiple EVM instructions. When the panel is displayed, it will be possible to step over every instruction, even those that refers to the same expression.

17.3.3 Solidity Locals

The Solidity Locals panel displays local variables associated with the current context.

17.3.4 Solidity State

The Solidity State panel displays state variables of the current executing contract.

17.3.5 Low level panels

These panels display low level informations about the execution:

- Stack
- Storages Changes
- Memory
- Call Data
- Call Stack
- Return Value (only if the current step is a RETURN opcode)
- Full Storages Changes (only at the end of the execution - display every storage change of every modified contract)

17.3.6 Reverted Transaction

A transaction can be `reverted` (because of an *out of gas exception* or Solidity `revert` statement or because of a low level exception).

It is important to be aware of the exception and to locate where the exception is in the source code.

Remix will warn you when the execution throws an exception. The `warning` button will jump to the last opcode before the exception happened.

17.3.7 Breakpoints

The two last buttons from the navigation area are used to jump either back to the previous breakpoint or forward to the next breakpoint.

Breakpoints can be added and removed by clicking on the line number in the *Editor*.

When using debug session with breakpoints, the execution will jump to the first encountered breakpoint.

Important note: If you add a breakpoint to a line that declares a variable, it might be triggered twice: Once for initializing the variable to zero and second time for assigning the actual value. As an example, assume you are debugging the following contract:

```
pragma solidity >=0.5.1 <0.6.0;

contract ctr {
    function hid () public {
        uint p = 45;
        uint m;
        m = 89;
        uint l = 34;
    }
}
```

And let's says that breakpoints are set for the lines

`uint p = 45;`

`m = 89;`

`uint l = 34;`

then clicking on `Jump to next breakpoint` will stop at the following lines in the given order:

`uint p = 45;` (declaration of p)

`uint l = 34;` (declaration of l)

`uint p = 45;` (45 assigned to p)

`m = 89;` (89 assigned to m)

`uint l = 34;` (34 assigned to l)

CHAPTER 18

Importing Source Files in Solidity

There are multiple techniques for importing files into Remix.

For a tutorial about importing files click here. You can also find this tutorial in the Remix Workshops plugin.

For a detailed explanation of the `import` keyword see the Solidity documentation

Here are a some of the main methods of importing a file:

18.1 Importing a file from the browser's local storage

Files in Remix can be imported with the `import` key word with the path to the file. Use `./` for relative paths to increase portability.

```
pragma solidity >=0.4.22 <0.6.0;

import "./ballot.sol";
```

18.2 Importing a file from your computer's filesystem

This method uses **remixd** - the remix daemon. Please go to the remixd tutorial for instructions about how to bridge the divide between the browser and your computers filesystem.

18.3 Importing from GitHub

It is possible to import files directly from GitHub. You should specify the release tag (where available), otherwise you will get the latest code in the master branch. For OpenZeppelin Contracts you should only use code published in an official release, the example below imports from OpenZeppelin Contracts v2.5.0.

```
pragma solidity >=0.4.22 <0.6.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.0/contracts/
↪math/SafeMath.sol";
```

18.4 Importing from Swarm

Files can be imported using all URLs supported by swarm. If you do not have a swarm node, then use swarm-gateways.net.

```
import 'bzz-raw://5766400e5d6d822f2029b827331b354c41e0b61f73440851dd0d06f603dd91e5';
```

18.5 Importing from IPFS

Files can be imported from IPFS.

```
import 'ipfs://Qmdyq9ZmWcaryd1mgGZ4PttRNctLGUSAMpPqufsk6uRMKh';
```

18.6 Importing from the console

You can also use a remix command remix.loadurl('<the_url>')in the console. You should specify the release tag (where available), otherwise you will get the latest code in the master branch. For OpenZeppelin Contracts you should only use code published in an official release, the example below imports from OpenZeppelin Contracts v2.5.0.

```
remix.loadurl('https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.0/
↪contracts/math/SafeMath.sol')
```

Notice that this will create a `github` folder in the file explorer. To load a file in the `github` folder, you would use a command like this:

```
import "github/OpenZeppelin/openzeppelin-contracts/contracts/math/SafeMath.sol";
```

CHAPTER 19

Remix Commands

In the console, you can run the commands listed below. Once you start to type a command, there is *auto completion*. These commands are using the following libraries:

- *ethers*: The ethers.js library is a compact and complete JavaScript library for Ethereum.
- *remix*: Ethereum IDE and tools for the web.
- *web3*: The web3.js library is a collection of modules which contain specific functionality for the ethereum ecosystem.
- *swarmgw*: This library can be used to upload/download files to Swarm via https://swarm-gateways.net/.

19.1 Here's the list of commands

remix.debug(hash): Start debugging a transaction.

remix.debugHelp(): Display help message for debugging

remix.execute(filepath): Run the script specified by file path. If filepath is empty, script currently displayed in the editor is executed.

remix.exeCurrent(): Run the script currently displayed in the editor.

remix.getFile(path): Returns the content of the file located at the given path

remix.help(): Display this help message.

remix.loadgist(id): Load a gist in the file explorer.

remix.loadurl(url): Load the given url in the file explorer. The url can be of type github, swarm or ipfs.

remix.setFile(path, content): set the content of the file located at the given path

remix.setproviderurl(url): Change the current provider to Web3 provider and set the url endpoint.

swarmgw.get(url, cb): Download files from Swarm via https**://swarm-gateways.net/

swarmgw.put(content, cb): Upload files to Swarm via https**://swarm-gateways.net/

ethers.Contract: This API provides a graceful connection to a contract deployed on the blockchain, simplifying calling and querying its functions and handling all the binary protocol and conversion as necessarily.

ethers.HDNode: A Hierarchical Deterministic Wallet represents a large tree of private keys which can reliably be reproduced from an initial seed.

ethers.Interface: The Interface Object is a meta-class that accepts a Solidity (or compatible) Application Binary Interface (ABI) and populates functions to deal with encoding and decoding the parameters to pass in and results returned.

ethers.providers: A Provider abstracts a connection to the Ethereum blockchain, for issuing queries and sending state changing transactions.

ethers.SigningKey: The SigningKey interface provides an abstraction around the secp256k1 elliptic curve cryptography library.

ethers.utils: The utility functions exposed in both the ethers umbrella package and the ethers-utils.

ethers.utils.AbiCoder: Create a new ABI Coder object

ethers.utils.RLP: This encoding method is used internally for several aspects of Ethereum, such as encoding transactions and determining contract addresses.

ethers.Wallet: A wallet manages a private/public key pair which is used to cryptographically sign transactions and prove ownership on the Ethereum network.

ethers.version: Contains the version of the ethers container object.

web3.bzz: Bzz module for interacting with the swarm network.

web3.eth: Eth module for interacting with the Ethereum network.

web3.eth.accounts: The web3.eth.accounts contains functions to generate Ethereum accounts and sign transactions and data.

web3.eth.abi: The web3.eth.abi functions let you de- and encode parameters to ABI (Application Binary Interface) for function calls to the EVM (Ethereum Virtual Machine).

web3.eth.ens: The web3.eth.ens functions let you interacting with ENS.

web3.eth.Iban: The web3.eth.Iban function lets convert Ethereum addresses from and to IBAN and BBAN.

web3.eth.net: Net module for interacting with network properties.

web3.eth.personal: Personal module for interacting with the Ethereum accounts.

web3.eth.subscribe: The web3.eth.subscribe function lets you subscribe to specific events in the blockchain.

web3.givenProvider: When using web3.js in an Ethereum compatible browser, it will set with the current native provider by that browser. Will return the given provider by the (browser) environment, otherwise null.

web3.modules: Contains the version of the web3 container object.

web3.providers: Contains the current available providers.

web3.shh: Shh module for interacting with the whisper protocol

web3.utils: This package provides utility functions for Ethereum dapps and other **web3.js packages.

web3.version: Contains the version of the web3 container object.

web3.eth.clearSubscriptions();: Resets subscriptions.

web3.eth.Contract(jsonInterface[, address][, options]): The **web3.eth.Contract object makes it easy to interact with smart contracts on the ethereum blockchain.

web3.eth.accounts.create([entropy]);: The web3.eth.accounts contains functions to generate Ethereum accounts and sign transactions and data.

CHAPTER 20

Remixd: Access your Local Filesystem

To give the Remix-ide (the web app) access to a folder on your local computer, you need to use `remixd`.

`remixd` is both the name of an npm module and the name of a Remix-plugin. You need to install the plugin (from the plugin manager) and you need to install the `remixd` npm module.

NOTE: you need to install the remixd npm module & Run its command before activating the remixd plugin.

The code of `remixd` is here .

20.1 remixd Installation

`remixd` can be globally installed using the following command: `npm install -g remixd`

Or just install it in the directory of your choice by removing the -g flag: `npm install remixd`

20.2 remixd Command

From the terminal, the command `remixd -s <absolute-path-to-the-shared-folder> --remix-ide <your-remix-ide-URL-instance>` will start `remixd` and will share the given folder with remix-ide.

For example, to use remixd with Remix IDE, use this command: `remixd -s <absolute-path-to-the-shared-folder> --remix-ide https://remix.ethereum.org`

Make sure that if you use https://remix.ethereum.org (**secure http**) in the remixd command (like in the example above), that you are also pointing your browser to https://remix.ethereum.org and not to http://remix.ethereum.org (plain old insecure http). Or if you want to use http in the browser use http in the remixd command.

The folder is shared using **a websocket connection** between `Remix IDE` and `remixd`.

Be sure the user executing `remixd` has read/write permission on the folder.

There is an option to run remixd in read-only mode, use `--read-only` flag.

20.3 Warning!

`remixd` provides `full read and write access` to the given folder for `any application` that can access the `TCP port 65520` on your local host.

20.4 After the command is running, activate the remixd plugin.

From `Remix IDE`, in the Plugin Manager, activate the remixd plugin. This plugin is a **websocket plugin** and it has no UI other than a modal dialog box.

This modal will ask confirmation

Accepting this dialog will start a session.

If you do not have `remixd` running in the background - another modal will open up and it will say:

```
Cannot connect to the remixd daemon.
Please make sure you have the remixd running in the background.
```

Assuming you don't get the 2nd modal, your connection to the remixd daemon is successful. The shared folder will be available in the file explorer.

When you click the activation of remixd is successful - there will NOT be an icon that loads in the icon panel.

Click the File Explorers icon and in the swap panel you should now see the folder for `localhost`.

Click on the `localhost connection` icon:

CHAPTER 21

FAQ

21.1 Solidity compiler

Q: Error: compiler might be in a non-sane state

```
error: "Uncaught JavaScript exception: RangeError: Maximum call stack size exceeded.
The compiler might be in a non-sane state, please be careful and do not use further␣
↪compilation data to deploy to mainnet.
It is heavily recommended to use another browser not affected by this issue (Firefox␣
↪is known to not be affected)."
```

A: Old versions of solidity compiler had this problem with chrome. Please change the compiler version in Solidity Plugin to the newer one or use another browser.

Q: I'm getting an issue with Maximum call stack exceed and various other errors, can't compile.

A: Try a different browser or a newer solidity compiler version.

Q: How to verify a contract that imports other contracts?

A: The verification tool does not recursively go through the import statments in a contract. So can only verify a 'flattened' contract.

There is a plugin called `Flattener` which will stuff all the original code and the imported code into a single file.

21.2 Deploy & Run

Q: I am using an Infura endpoint in my app, but when I try to deploy against that endpoint in remix IDE selecting "web3 provider" and putting my endpoint in, it's telling me that it can't connect

A: If the endpoint you are using is http, it won't work.

Q: Where is deploy button?

A: Its in the Deploy & Run module. If you haven't activated that module, you should do that by clicking Deploy & Run module in the Plugin Manager. You could also activate everything you need to work with solidity on the landing page (click the remix logo at the top left for the screen) and click the "Solidity" button in the environment section.

Q: How to pass a tuple to a public function in Remix?

A: Pass it as an array [].

Q: How to input a struct as input to a parameter of a function in the Deploy & Run module?

A: For inputting a struct, just like a tuple, pass it in as an array []. Also you need to put in the line:

`pragma experimental ABIEncoderV2;` at the top of the solidity file.

For example, here's a solidity file with a struct is an input parameter.

```
pragma solidity >=0.4.22 <0.6.0;
pragma experimental ABIEncoderV2;

contract daPeeps {
    struct Peep {uint a; uint b;} // declaration of Peep type
    Peep peep; //declaration of an object of Peep type

    constructor () public
    {
        peep.a = 0; // definition/initialisation of object
        peep.b = 0; //
    }

    function initPeepToPeep(Peep memory i) public payable {
        peep.a = i.a;
        peep.b = i.b;
    }
    function setPeep(uint a, uint b) public payable {
        peep.a = a;
        peep.b = b;
    }

    function getPeep() public view returns(Peep memory)
    {
        return peep;
    }
}
```

The input of initPeepToPeeps takes a struct. If you input `[1,2]` the transaction will go through.

21.3 General

Q: Where do plugin developers go with their questions?

A: The Gitter Remix plugin developers room https://gitter.im/ethereum/remix-dev-plugin

CHAPTER 22

Remix URLs & Links with Parameters

22.1 Remix URLs

- An online version is available at https://remix.ethereum.org. This version is stable and is updated at almost every release.
- An alpha online version is available at https://remix-alpha.ethereum.org. This is not a stable version.
- Github repo: https://github.com/ethereum/remix-project . The README contains instructions for running Remix-IDE locally.
- Github release: https://github.com/ethereum/remix-project/releases .

22.2 Embedding & Linking to Remix

Remix-IDE's urls have parameters -so it is possible to specify:

- the list of plugins you want activated
- the theme (Dark or Light)
- the panels that should be minimized
- if you want the Solidity compiler to have **optimize** enabled

In the following example, there is a list of plugins that follows the word **plugins** will be activated and the last plugin will gain the focus.

```
https://remix.ethereum.org/?#activate=solidity,solidityUnitTesting,udapp,defiexplorer
```

For the plugin are called by their name in their profile. To check for a plugin's profile name - for plugins built by external teams, please go to https://github.com/ethereum/remix-plugins-directory/tree/master/plugins

22.3 Further Customization with URL parameters

The following URL will close everything except the main panel & the icon panel (so the side and terminal are minimized)

https://remix.ethereum.org/?#embed=true

To link with the side panel minimized use this URL:

https://remix.ethereum.org/?#minimizesidepanel=true

To link to Remix with the dark theme or the light theme specified use this url:

https://remix.ethereum.org/?#theme=Dark

To link to Remix with the Solidity compiler, the unit testing, and LearnEth plugins activated (with Learneth gaining the side panel's focus) & with the Light theme loaded & with the terminal minimized use this URL & with optimize off:

https://remix.ethereum.org/?#activate=solidity,solidityUnitTesting,LearnEth&theme=Light&minimizeterminal=true&optimize=false&e
v0.6.6+commit.6c089d02.js

CHAPTER 23

Remix Github Tutorials

There are a series of tutorials in our github repo remix-workshops.

We are in the process of upgrading these tutorials to use the new Remix layout.

In this repo there tutorials for all levels.

There are tutorials for specific remix functionalities like:

Deploying

```
Multiple ways of loading files in Remix
Deploying with libraries
Deploying a proxy contract
```

Testing

```
Testing Examples
Continuous integration
```

Remix Plugin Development

```
Developing a plugin for Remix and deploying it to swarm
```

Other

```
EtherAtom (walkthrough slides + screencast)
Debugging transactions with Remix IDE
Recording and replaying transactions
Using a Pipeline plugin for developing Solidity contracts with demo video
Running scripts in the Remix terminal (batch deployment) (proxy deployment)
```

Additional external workshops

```
Using Oraclize plugin in Remix
```

CHAPTER 24

Code Contribution Guide

Remix is an open source tool and we encourage everyone to help us improve it. Please opening issues, give feedback or contribute by a pulling request to our codebase.

The Remix application is built with JavaScript and it doesn't use any frameworks. We rely on a selected set of npm modules, like `yo-yo`, `csjs-inject` and among others. Check out the `package.json` files in the Remix submodules to learn more about the stack.

To learn more, please visit our GitHub page.

CHAPTER 25

Community Support

We know that blockchain ecosystem is very new and that lots of information is scattered around the web. That is why we created a community support channel where we and other users try to answer your questions if you get stuck using Remix. Please, join the community and ask for help.

For anyone who is interested in developing a custom plugin for Remix or who wants to contribute to the codebase, we opened a contributors' channel especially for developers working on Remix tools.

We would kindly ask you to respect the space and to use it for getting help with your work and the developers' channel for discussions related to working on Remix codebase. If you have ideas for collaborations or you want to promote your project, try to find some more appropriate channels to do so. Or you can contact the main contributors directly on Gitter or Twitter.