Remix Documentation

Release 1

yann300

May 20, 2019

New Layout Intro

1 Remix-IDE Layout **3**
- 1.1 The new structure . . . 3
- 1.2 Icon Panel at Page Load . . . 4
- 1.3 Homepage . . . 5
- 1.4 Plugin Manager . . . 5
- 1.5 Themes . . . 6

2 File Explorer **7**
- 2.1 Create new File . . . 8
- 2.2 Add Local File . . . 8
- 2.3 Publish to Gist . . . 8
- 2.4 Copy to another Remix instance . . . 8
- 2.5 Connect your filesystem to Remix . . . 8

3 Plugin Manager **9**

4 Settings **11**

5 Solidity Editor **13**

6 Terminal **15**

7 Compiler (Solidity) **17**

8 Run & Deploy **21**
- 8.1 Run Setup . . . 22
- 8.2 Initiate Instance . . . 23
- 8.3 Pending Instances . . . 23
- 8.4 Using the ABI . . . 24
- 8.5 Using the Recorder . . . 24

9 Run & Deploy (part 2) **29**
- 9.1 Deployed contracts . . . 29

10 Debugger **31**

11 Analysis **33**

12 Build Artifact **37**
12.1 Library Deployment . . . 37

13 Creating and Deploying a Contract **39**
13.1 Selecting the VM mode . . . 39
13.2 Sample contract . . . 39
13.3 Deploying an instance . . . 40
13.4 Interacting with an instance . . . 42

14 Importing Source Files in Solidity **45**
14.1 Importing a file from the browser's local storage . . . 45
14.2 Importing a file from your computer's filesystem . . . 45
14.3 Importing from GitHub . . . 46
14.4 Importing from Swarm . . . 46

15 Remixd: Get access your local filesystem **47**

16 Unit Testing **49**
16.1 Generate test File . . . 50
16.2 Run Tests . . . 50
16.3 Continuous integration . . . 50

17 Remix Plugin **51**

18 Finding Remix **53**

19 Remix Tutorials **55**

20 Code contribution guide **57**

21 Community **59**

22 Support chat **61**

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.

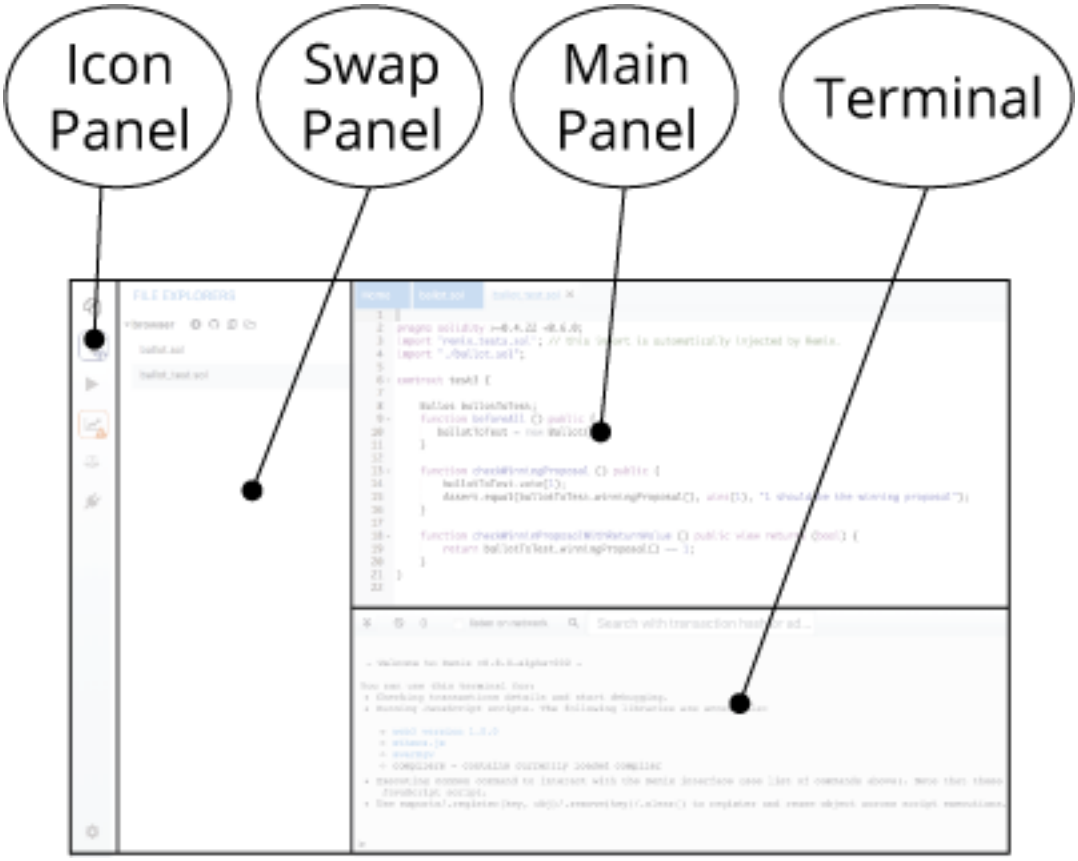
Useful links:

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

CHAPTER 1

Remix-IDE Layout

1.1 The new structure

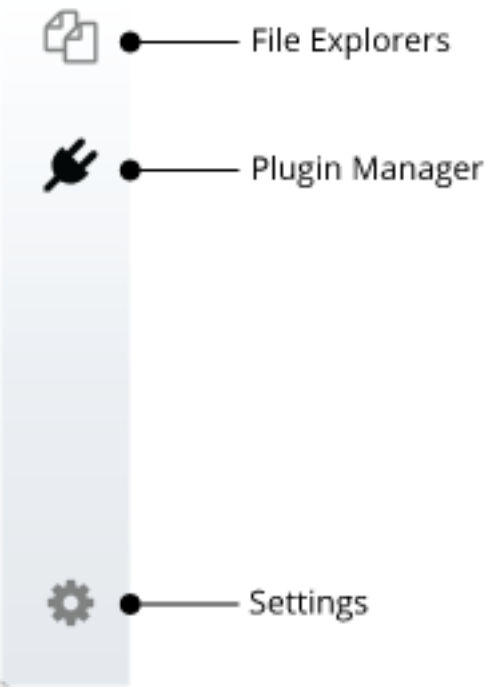


1. Icon Panel - click to change which plugin appears in the Swap Panel

2. Swap 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

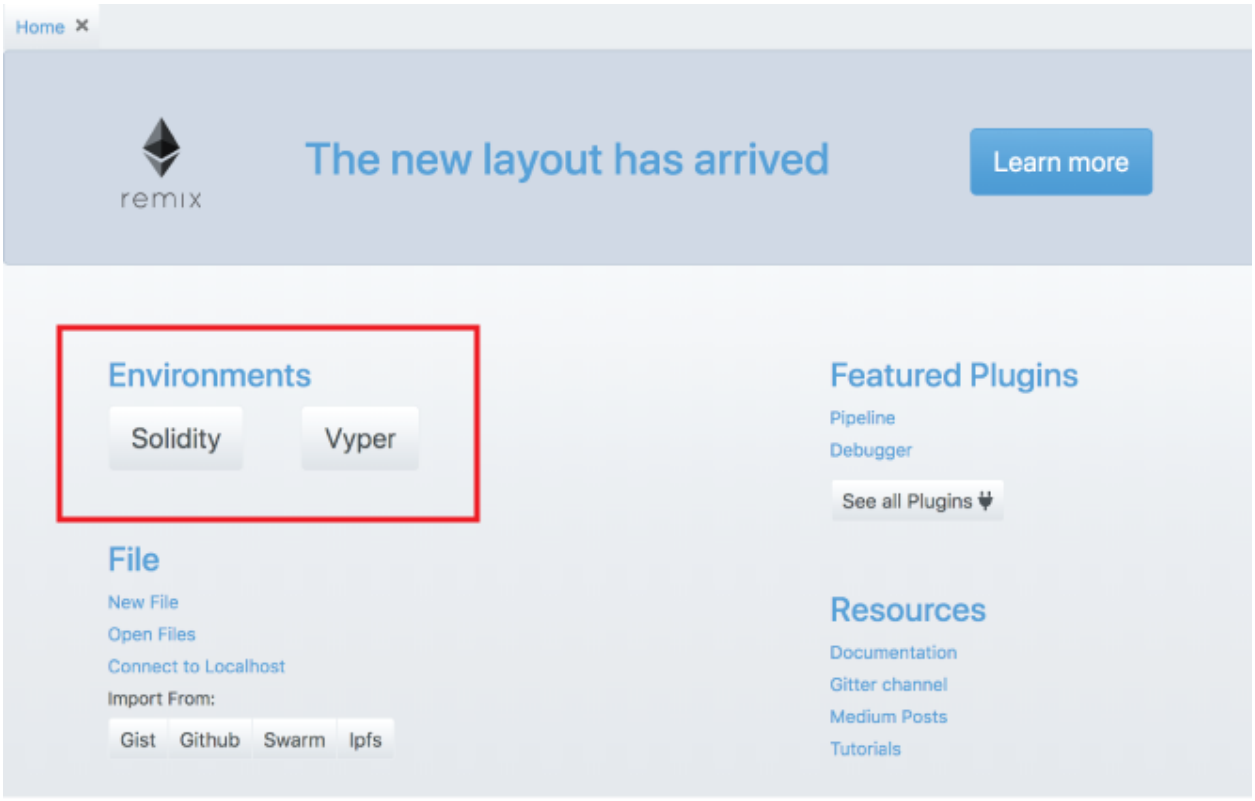
When you load remix - the icon panel show these icons by default.



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.

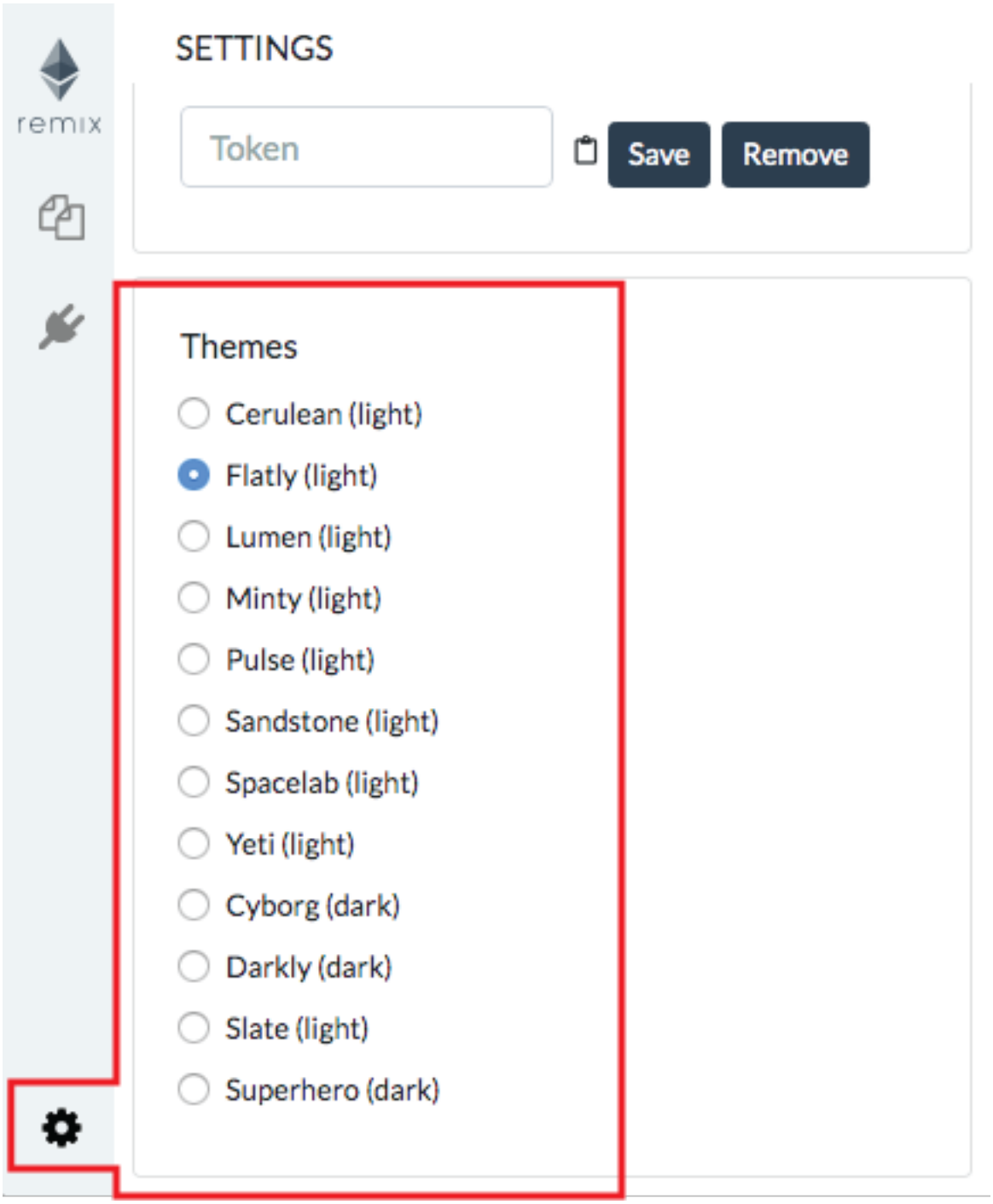
1.4 Plugin Manager

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

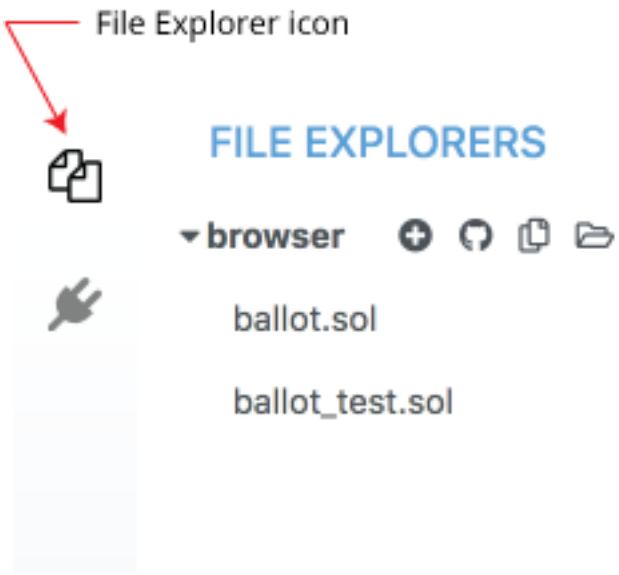
So you want to work on Remix with a dark theme or a gray theme or just a different theme that the one you are currently looking at? Go to the settings tab and at the bottom is a choice of lots of bootstrap based themes.



CHAPTER 2

File Explorer

To get to the file explorers - click the file explorers icon.



The file explorer lists by default all the files stored in your browser. You can see them in the browser folder. You can always rename, remove or add new files to the file explorer.

Note that clearing the browser storage will permanently delete all the solidity files you wrote. To avoid this, you can use Remixd, which enables you to store and sync files in the browser with your local computer (for more information see remixd).

We will start by reviewing at the icons at the top left - from left to the right:

2.1 Create new File

Creates a new `untitled.sol` file in Remix.

2.2 Add Local File

Allows you to select files from the local file system and import them to the Remix browser storage.

2.3 Publish to Gist

Publishes all files from the browser folder to a gist. Gist API has changed in 2018 and it unfortunately 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 only Create gists checkbox and generate a new token.

Then paste it in Remix (right panel/Settings tab) and click Save. Now you should be able to use the feature.

2.4 Copy to another Remix instance

Enables you to copy files from the browser storage to another instance (URL) of Remix.

2.5 Connect your filesystem to Remix

Allows to sync between Remix and your local file system (see more about RemixD).

CHAPTER 3

Plugin Manager

** This is text copied from layout.md - it needs an image & more info about connect to the local plugin

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.

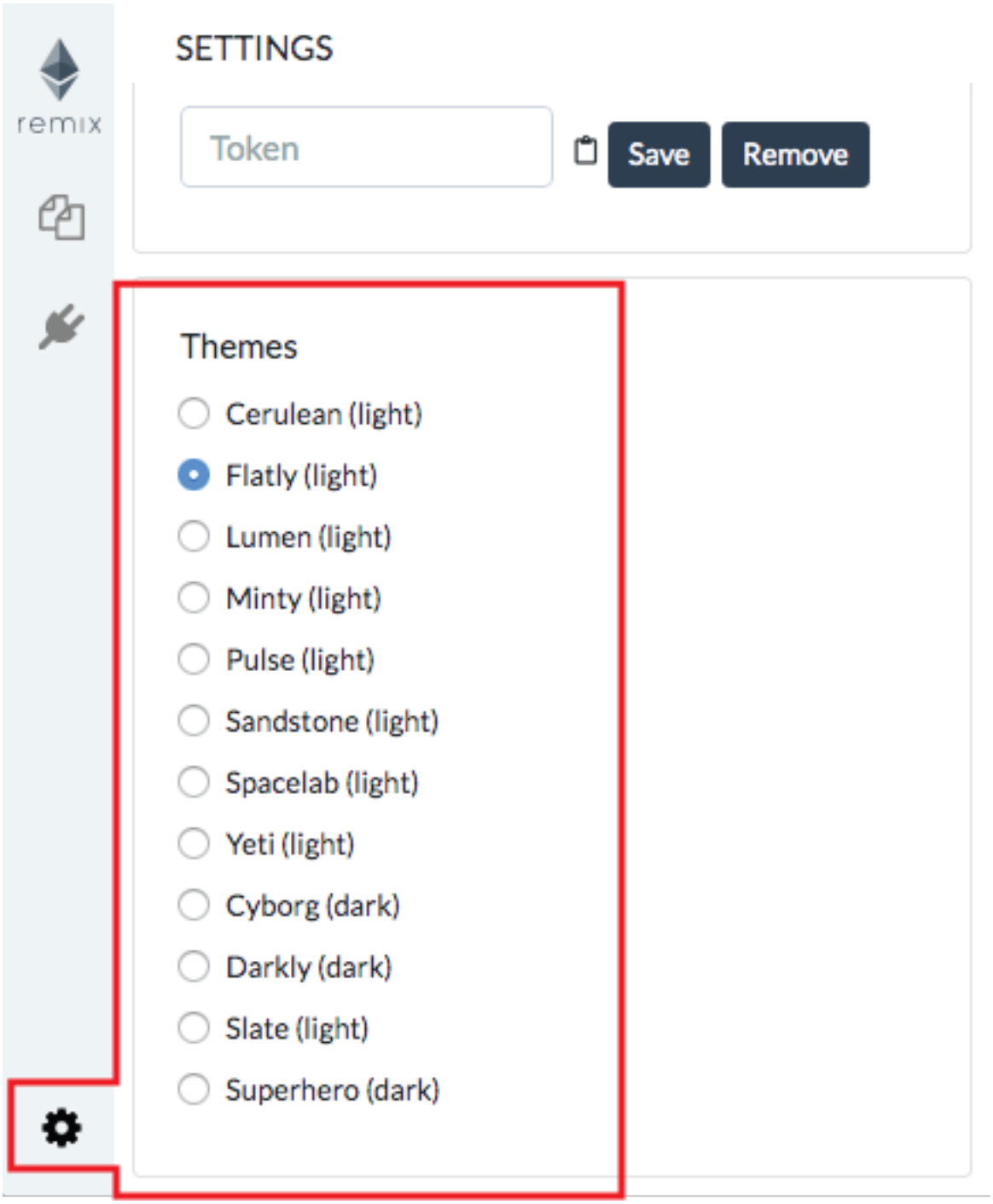
The Plugin Manager is also the place you go when you are creating your own plugin and you want to load your local plugin into Remix. In that case you'd click on the "Connect to a Local Plugin" link at the top of the Plugin Manager panel.

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.



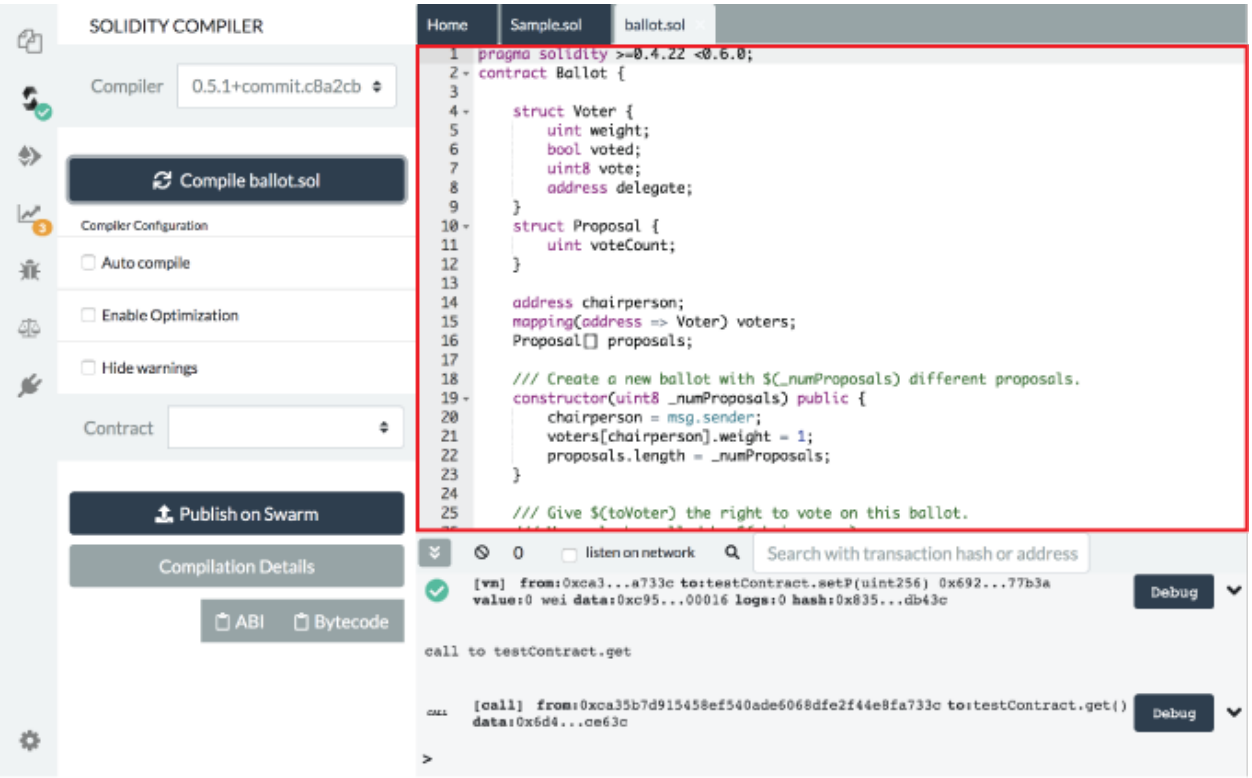
Another important settings:

- 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.



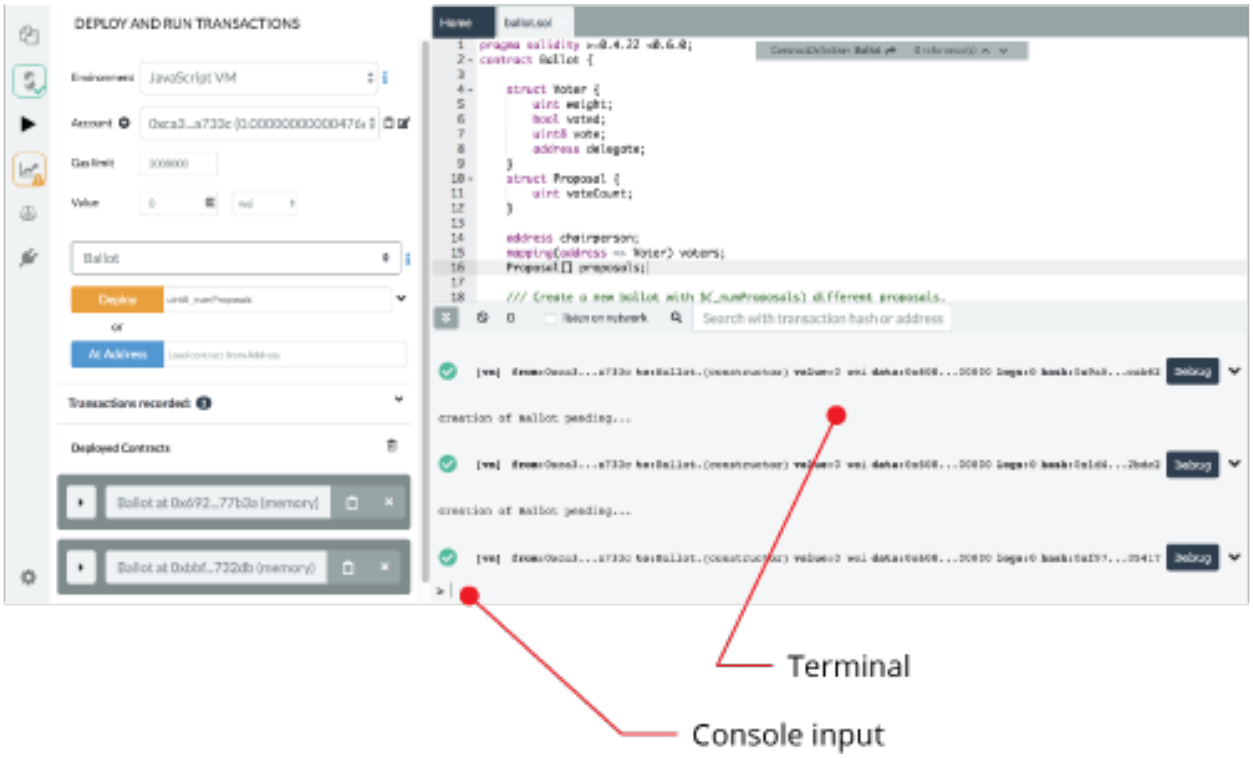
Here's the list of some important features:

- 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 >.

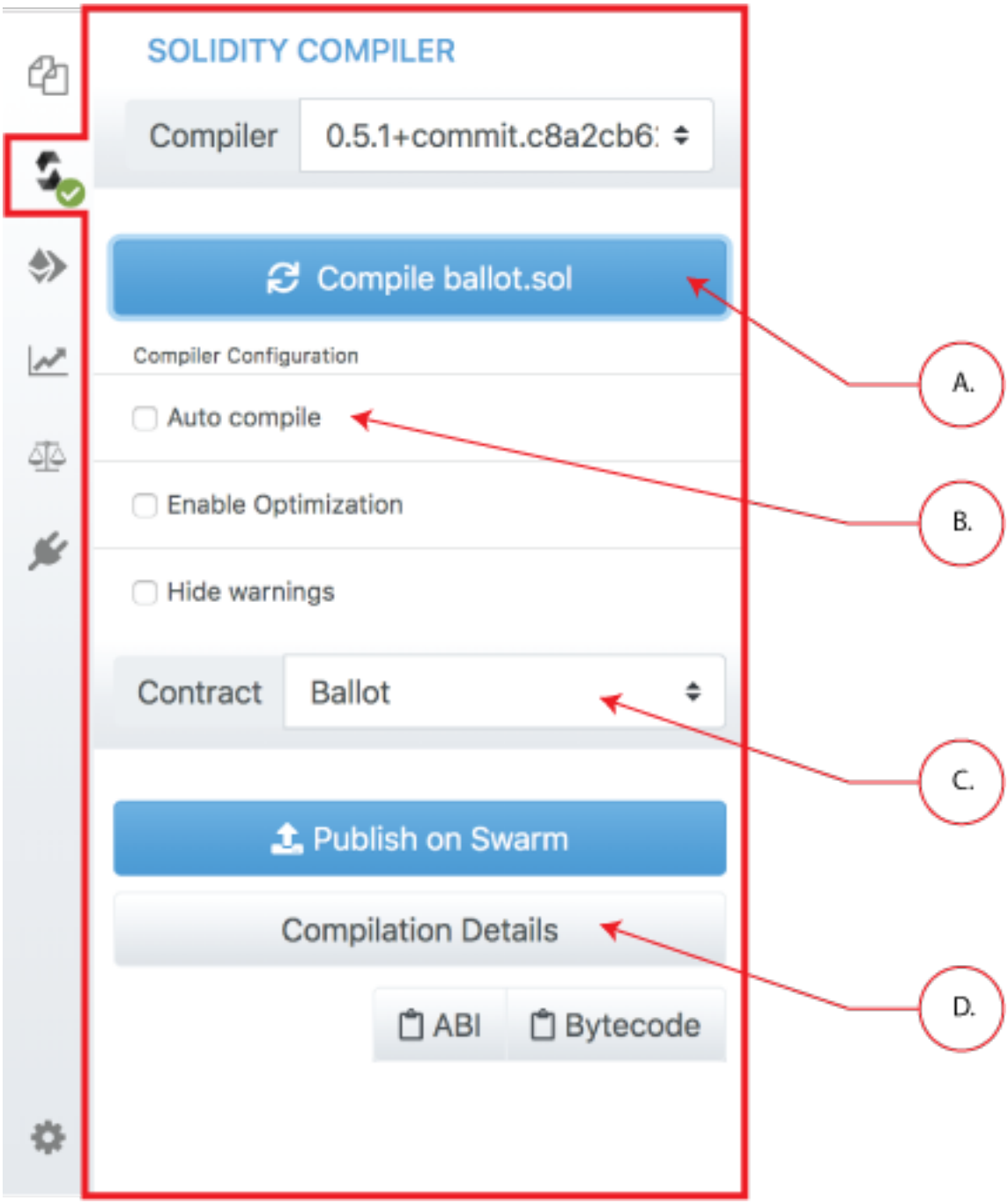
CHAPTER 7

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 (**A. 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 (**B. in image below**).

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 the newly compiled contracts. The contract compiled can be selected with the Contract pulldown menu (**C. in image below**). 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 (**D. in image below**), a modal opens displaying detailed information about the current selected contract.

From this tab, you can also publish your contract to Swarm (only non abstract contracts can be published).

Published data notably contains the `abi` and 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 very valuable when addressing reported issues even if the compiler doesn't complain. (see more)

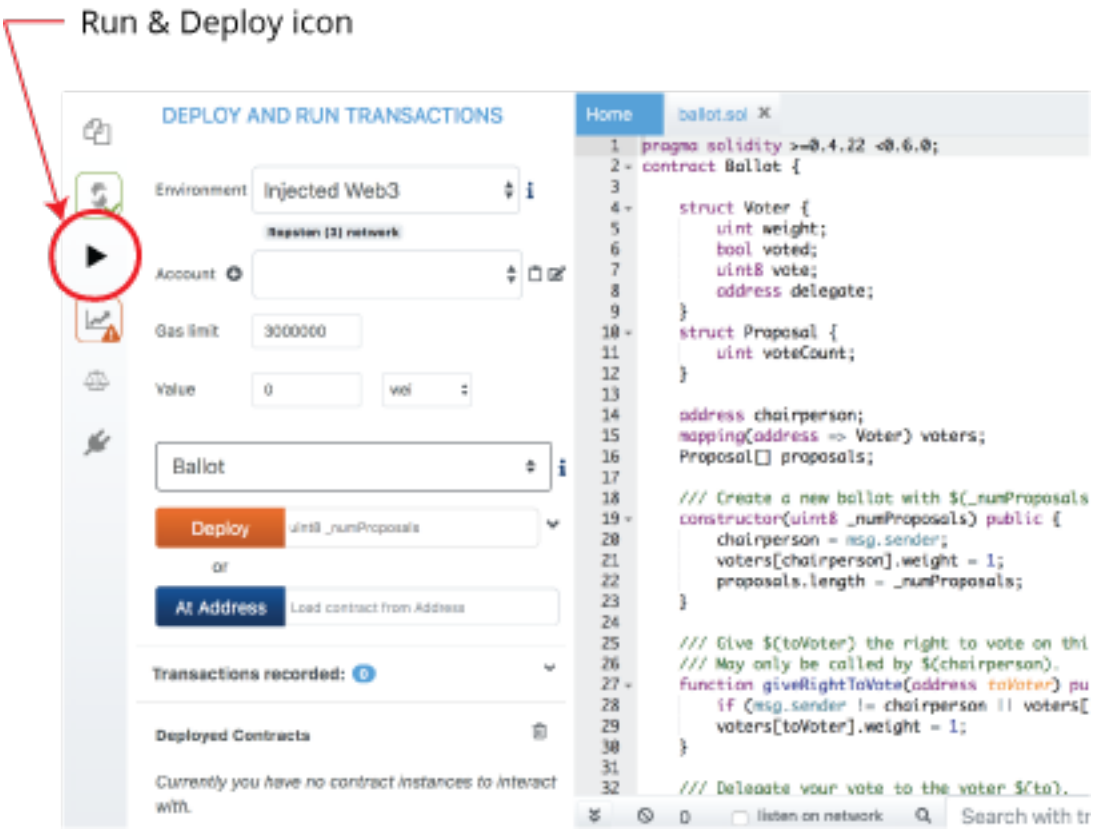
CHAPTER 8

Run & Deploy

The Run tab allows you to send transactions to the current environment.

To get to the Run & Deploy module - click the run icon in the icon panel.

In order to use this module you need to have a contract compiled. So if there is file name in the contract pulldown menu (in the image below it's the pulldown that says **Ballot**), then you can interact with this contract. If nothing is there - then you need to select a contract - make it the active contract in the main panel, (in the image below - on the right side of the page - in the main panel - you see the ballot.sol so it is the active contract) then go to the compiler module and compile it.

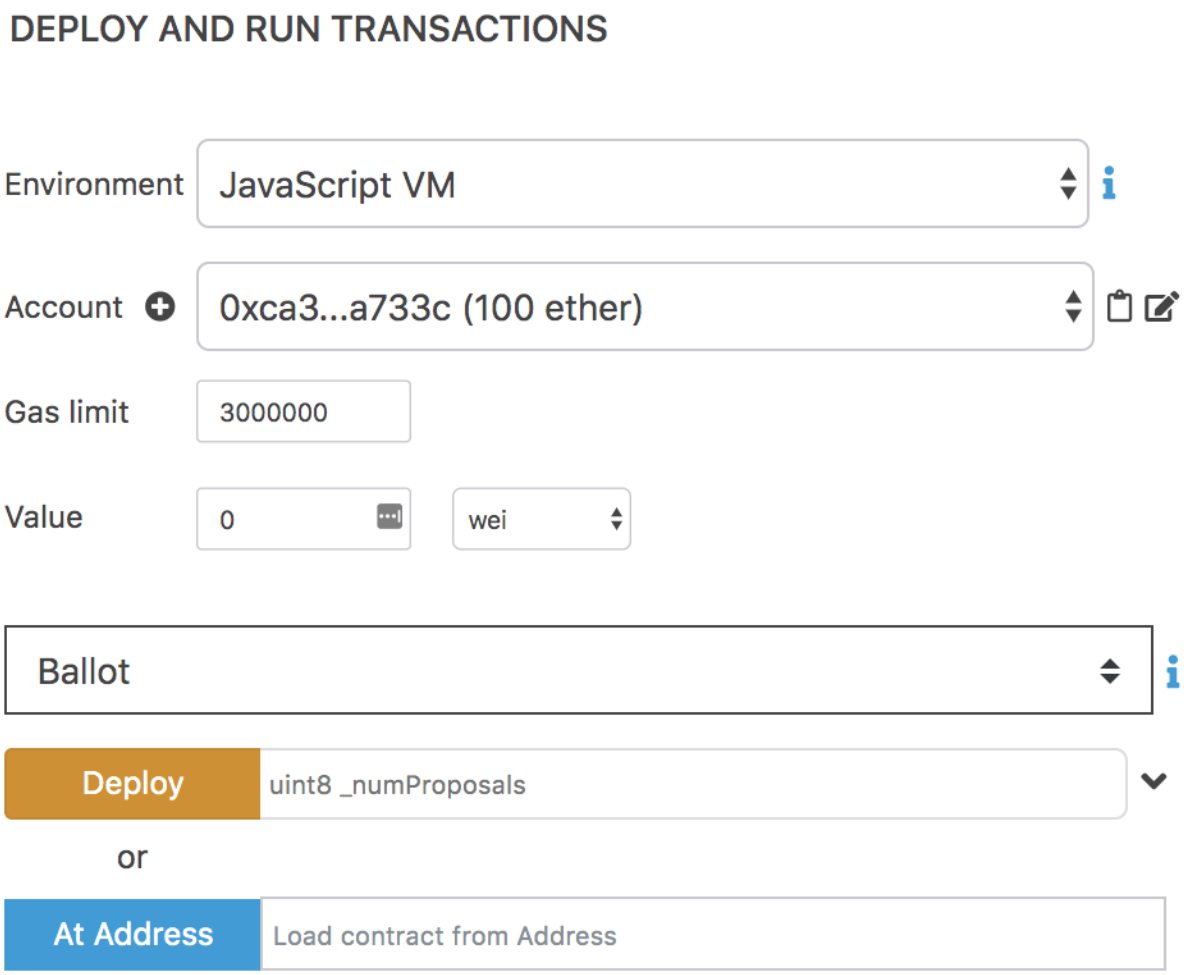


8.1 Run Setup

The following settings allow you to directly influence the transaction execution:

Environment:

- `JavaScript VM`: All the transactions will be executed in a sandbox blockchain in the browser. This means nothing will be persisted and a page reload will restart a new blockchain from scratch, the old one will not be saved.
- `Injected Provider`: Remix will connect to an injected web3 provider. `Metamask` is an example of providers that inject web3, thus can be used with this option.
- `Web3 Provider`: Remix will connect to a remote node. You will need to provide the URL address to the selected provider: geth, parity or any Ethereum client.
- Account: the list of accounts associated with the current environment (and their associated balances).
- Gas Limit: the maximum amount of gas that can be set for all the transactions created in Remix.
- Value: the amount of value for the next created transaction (this value is always reset to 0 after each transaction execution).



8.2 Initiate Instance

This section contains the list of compiled contracts and 2 actions:

- `At Address` assumes the given address is an instance of the selected contract. It is then possible to interact with an already deployed contract. There's no check at this point, so be careful when using this feature, and be sure you trust the contract at that address.
- `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.

8.3 Pending Instances

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

8.4 Using the ABI

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

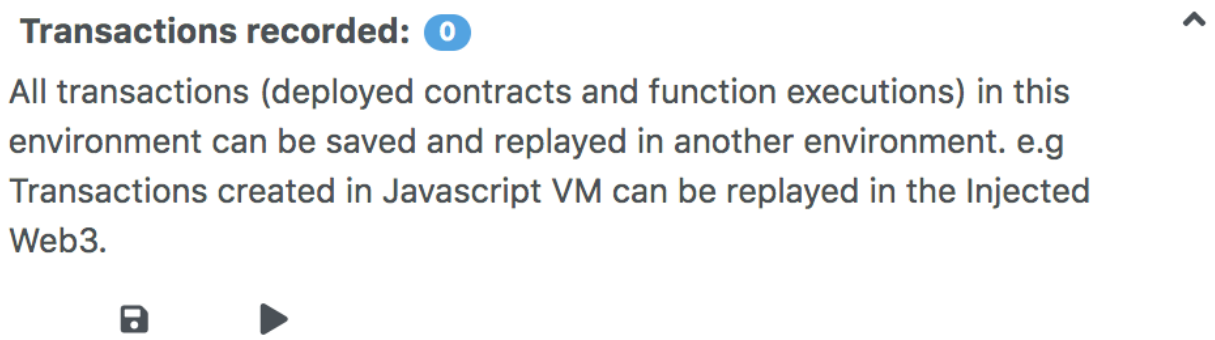
To interact with a contract using the ABI, create a new file in Remix with extension `*.abi` and copy the ABI content to it. Then in the input next to `At Address`, put the Address of the contract you want to interact with. Click on `At Address`, a new "connection" with the contract will popup below.

8.5 Using the Recorder

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

Saving to JSON allows to easily check the transaction list, tweak input parameters, change linked library, etc...

We can find many use cases for the recorder, for instance: : - After having coded and tested contracts in a constrained environment (like the JavaScript VM), it could be interesting to redeploy them easily in a more persisted environment (like a Geth node) in order to check whether everything behaves normally in a classic environment. - Deploying contract does often require more than creating one transaction. - Working in a dev environment does often require to setup the state in a first place.



Saving a record ends up with the creation of this type of content (see below):

In that specific record, 3 transactions are executed:

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

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

The third parameter 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}`.

```
{.sourceCode .none}
{
"accounts": {
    "account{0}": "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
```

(continues on next page)

(continued from previous page)


```
},
"linkReferences": {
    "testLib": "created{1512830014773}"
},
"transactions": [
    {
    "timestamp": 1512830014773,
    "record": {
        "value": "0",
        "parameters": [],
        "abi": "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a",
        "contractName": "testLib",
        "bytecode":
↪"60606040523415600e57600080fd5b60968061001c6000396000f300606060405260043610603f576000357c0100000000
↪",
        "linkReferences": {},
        "type": "constructor",
        "from": "account{0}"
    }
    },
    {
    "timestamp": 1512830015080,
    "record": {
        "value": "100",
        "parameters": [
        11
        ],
        "abi": "0xc41589e7559804ea4a2080dad19d876a024ccb05117835447d72ce08c1d020ec",
        "contractName": "test",
        "bytecode":
↪"60606040526040516020806102b1833981016040528080519060200190919050508060008190555505061027a806100376
↪_browser/ballot.sol:testLib____________
↪636d4ce63c60006040516020015260405181630fffffff167c0100000000000000000000000000000000000000000000000
↪",
        "linkReferences": {
        "browser/ballot.sol": {
            "testLib": [
            {
                "length": 20,
                "start": 511
            }
            ]
        }
        },
        "name": "",
        "type": "constructor",
        "from": "account{0}"
    }
    },
    {
    "timestamp": 1512830034180,
    "record": {
        "value": "1000000000000000000",
        "parameters": [
        1,
        "0xca35b7d915458ef540ade6068dfe2f44e8fa733c"
        ],
```


(continues on next page)

(continued from previous page)

```
        "to": "created{1512830015080}",
        "abi": "0xc41589e7559804ea4a2080dad19d876a024ccb05117835447d72ce08c1d020ec",
        "name": "set",
        "type": "function",
        "from": "account{0}"
    }
    }
],
"abis": {
    "0xbc36789e7a1e281436464229828f817d6612f7b477d66591ff96a9e064bcc98a": [
    {
        "constant": true,
        "inputs": [],
        "name": "get",
        "outputs": [
        {
            "name": "",
            "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": [],
```

(continues on next page)

(continued from previous page)

```
        "name": "getAddress",
        "outputs": [
        {
            "name": "",
            "type": "address"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "constant": false,
        "inputs": [
        {
            "name": "_t",
            "type": "uint256"
        },
        {
            "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).

Needs an Image

Several cases apply:

- The called function is declared as `constant` or `pure` in Solidity. The action has a blue background, clicking it does not create a new transaction. Clicking it is not necessary because there are not state changes - but it will update the return value of the function.
- The called function has no special keywords. The action has a light red background, clicking on does create a new transaction. But this transaction cannot accept any amount of Ether.
- The called function is declared as `payable` in Solidity. The action has a red background, clicking it does create a new transaction and this transaction can accept value.

For more information see more about Solidity modifier .

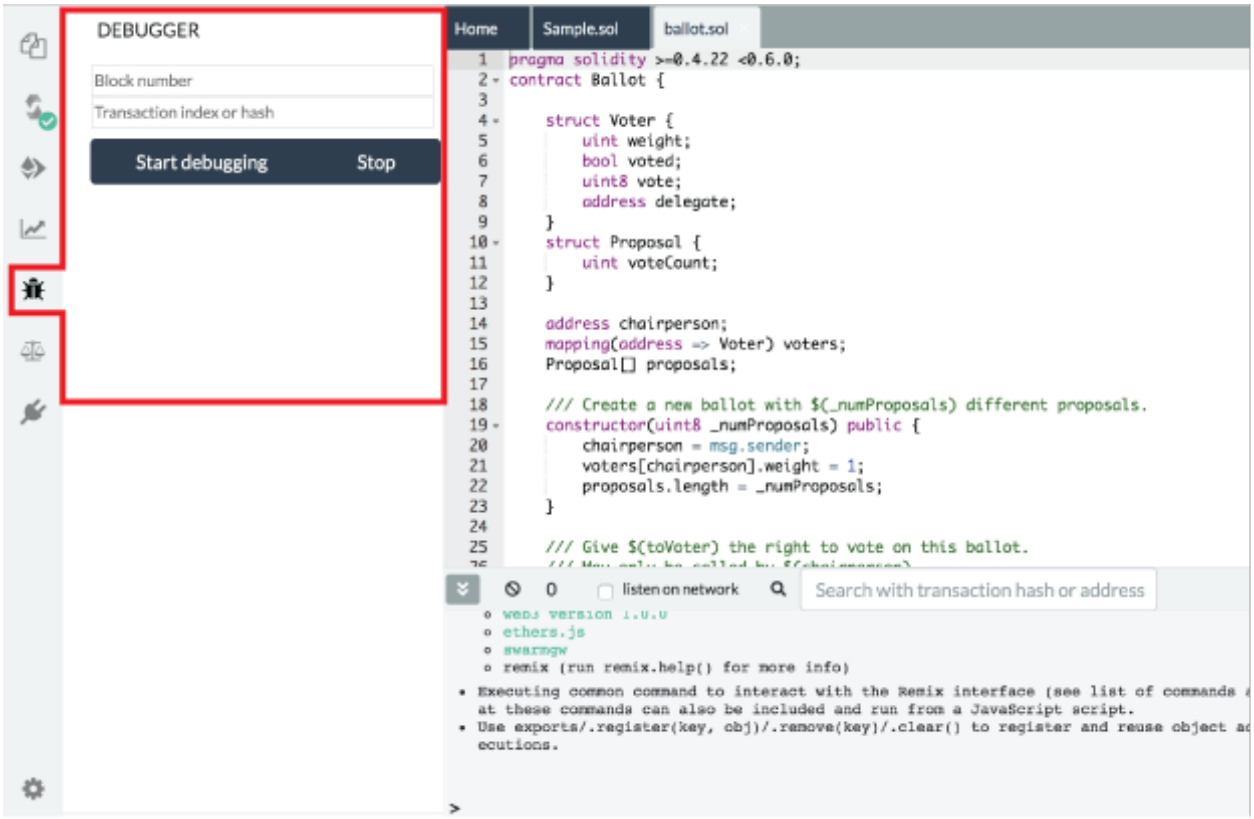
If a function requires input parameters, it is required to specify them.

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.

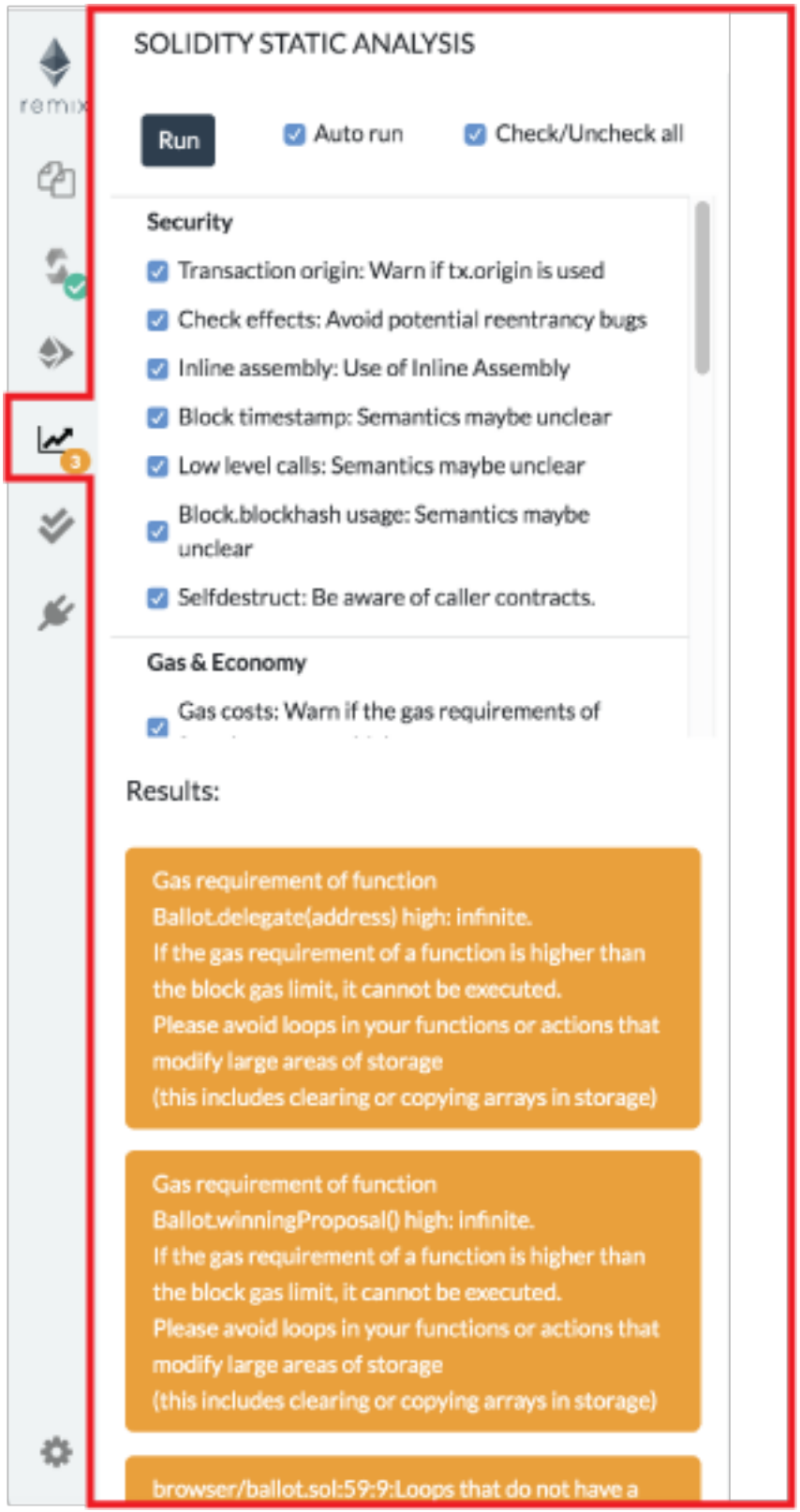


CHAPTER 11

Analysis

This section gives information about the last compilation. By default, a new analysis is run at each compilation.

The analysis tab gives detailed information about the contract code. It can help you avoid code mistakes and to enforce best practices.



Here is the list of analyzers:

Security:

- Transaction origin: Warns if tx.origin is used
- Check effects: Avoid potential reentrancy bugs
- Inline assembly: Use of Inline Assembly
- Block timestamp: Semantics maybe unclear
- Low level calls: Semantics maybe unclear
- Block.blockhash usage: Semantics maybe unclear

Gas & Economy:

- Gas costs: Warns if the gas requirements of the functions are too high
- This on local calls: Invocation of local functions via this

Miscellaneous:

- Constant functions: Checks for potentially constant functions
- Similar variable names: Checks if variable names are too similar

CHAPTER 12

Build Artifact

As compilation succeed Remix create a JSON file for each compiled contract. These JSON files contains several metadata

12.1 Library Deployment

By default Remix automatically deploy needed libraries.

`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 follow the form: `<network_name>:<networkd_id>`, but it is also possible to define `<network_name>` or `<network_id>` as keys.

```
{
        "VM:-": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
                },
                "autoDeployLib": true
        },
        "main:1": {
                "linkReferences": {
                        "browser/Untitled.sol": {
                                "lib": "<address>",
                                "lib2": "<address>"
                        }
```

(continues on next page)

(continued from previous page)

```
                },
                "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
        }
}
```

CHAPTER 13

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.

13.1 Selecting the VM mode

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

13.2 Sample contract

```
{.sourceCode .none}
pragma solidity ^0.5.1;

contract testContract {

    uint value;

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

(continues on next page)

(continued from previous page)

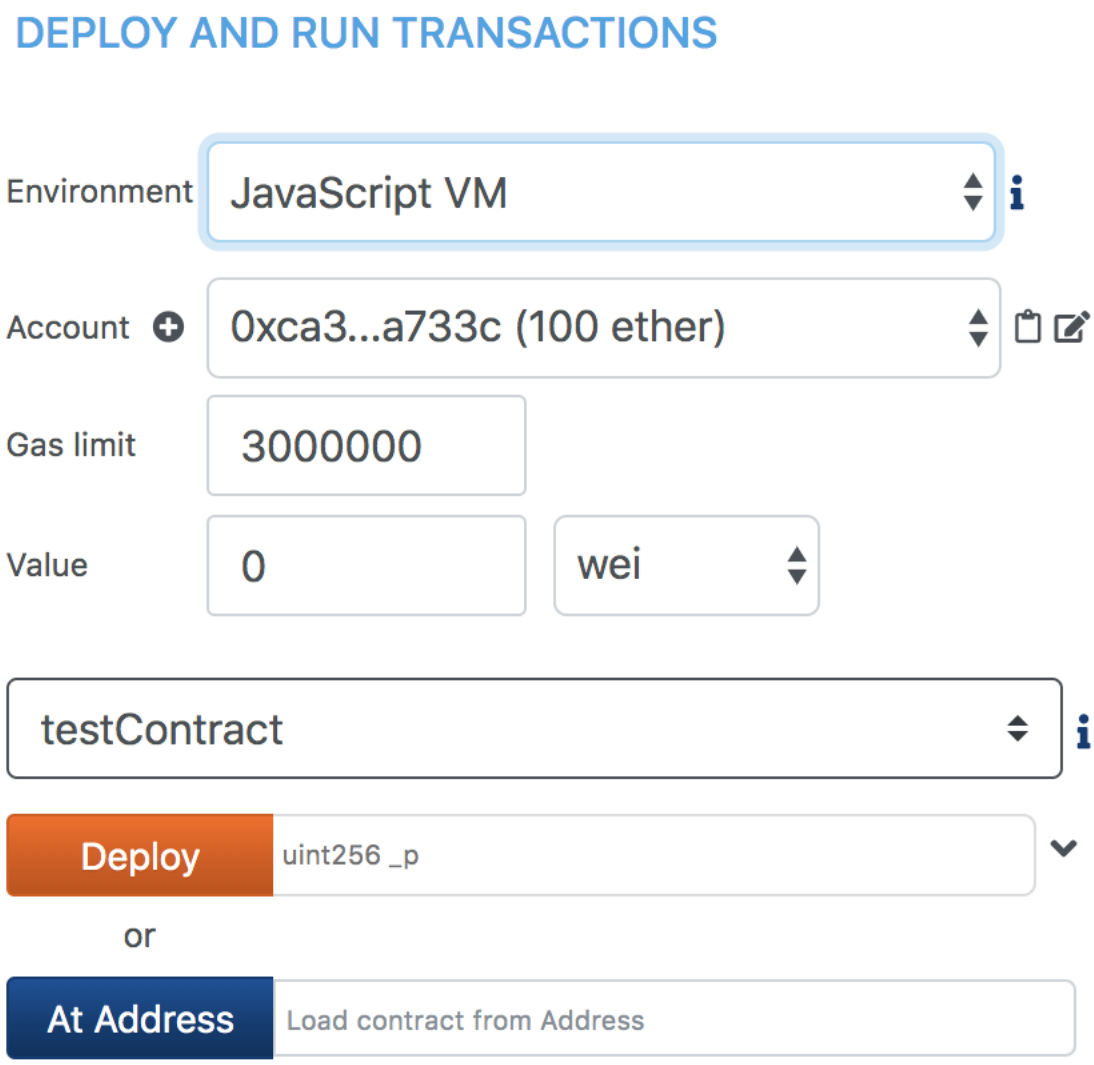
```
    }

    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.

13.3 Deploying an instance

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

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.



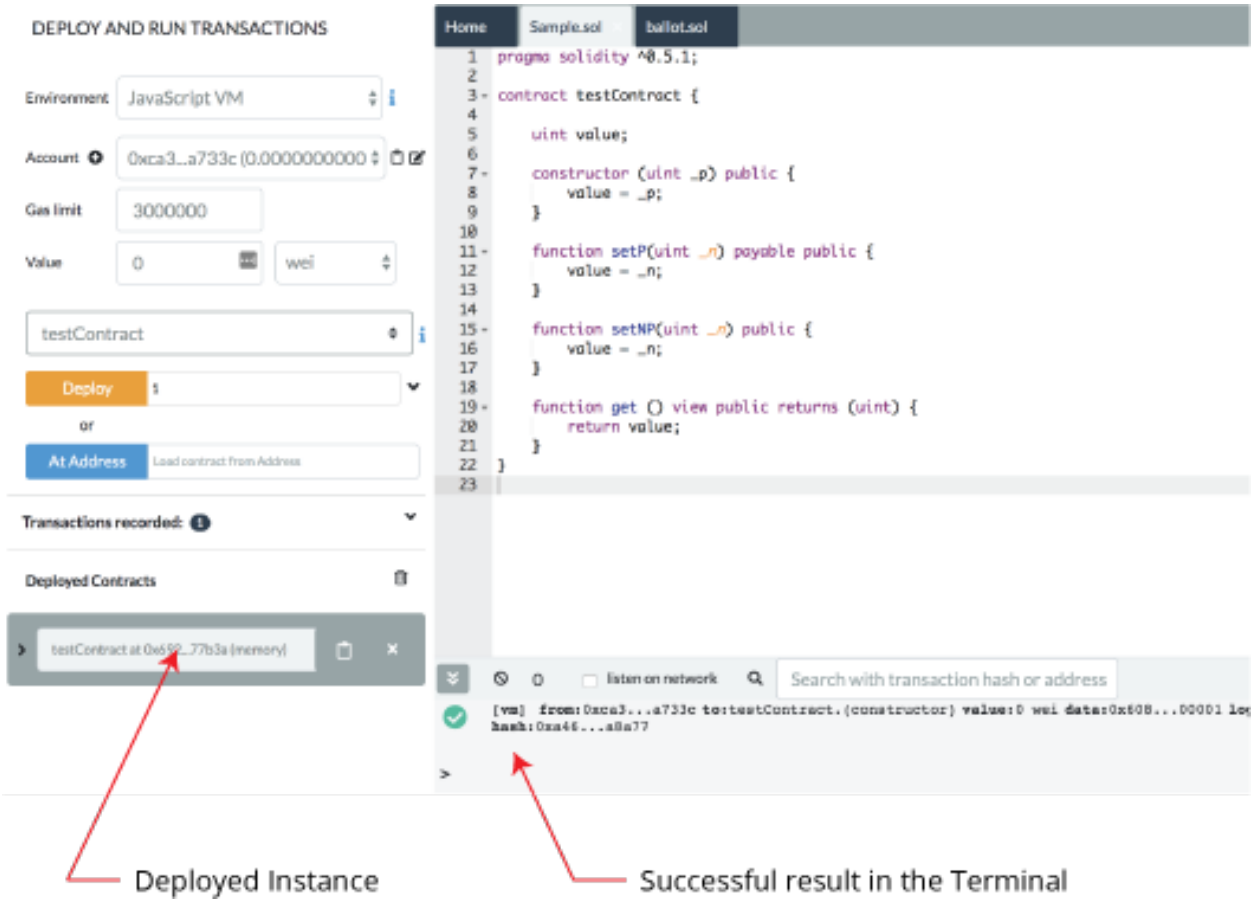
The constructor of `Ballot.sol` needs a parameter (of type `uint8`). Give any value and click on `Deploy`.

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`.



13.4 Interacting with an instance

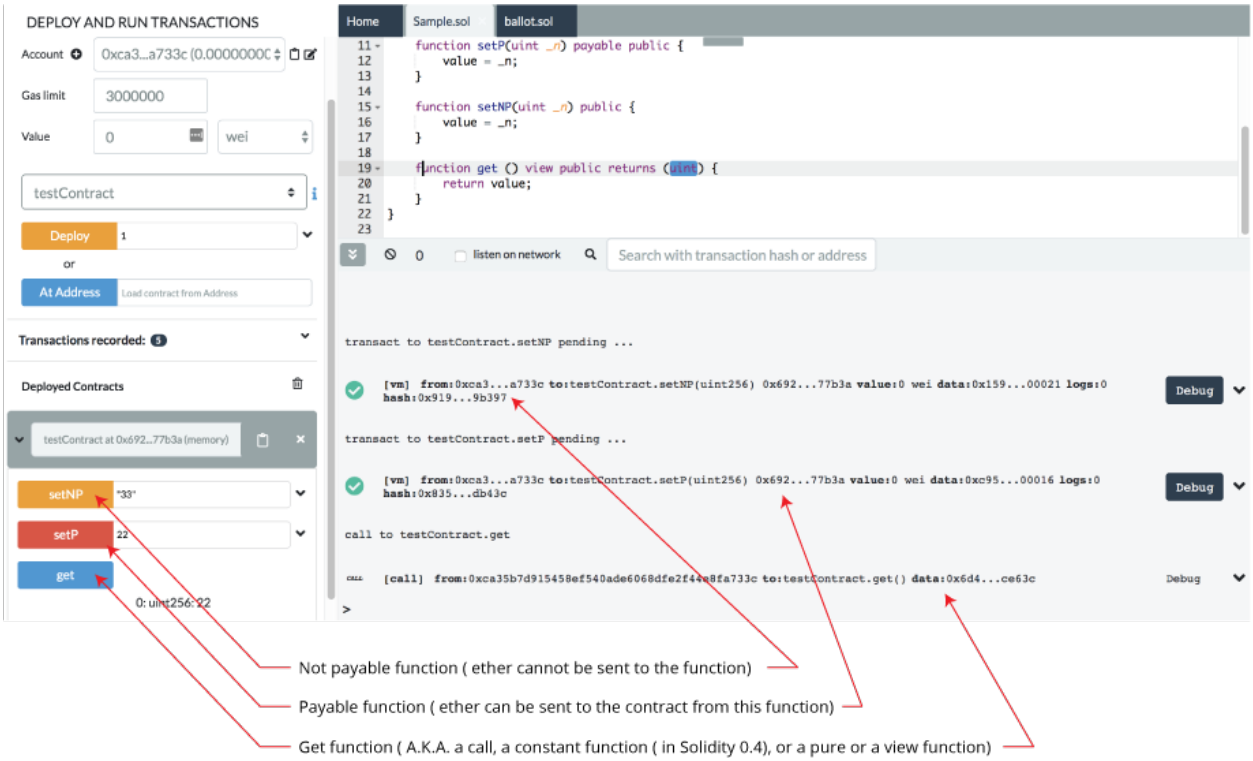
This new instance contains 3 actions which corresponds to the 3 functions (`setP`, `setPN`, `get`). Clicking on `SetP` or `SetPN` will create a new transaction.

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.


DEPLOY AND RUN TRANSACTIONS
Account
Gas limit
3000000
Value
wei
testContract
Deploy
or
At Address
Transactions recorded:
Deployed Contracts
setNP
setP
get
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 ...
transact to testContract.setP pending ...
call to testContract.get
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 14

Importing Source Files in Solidity

It is essential to know all many techniques for importing files.

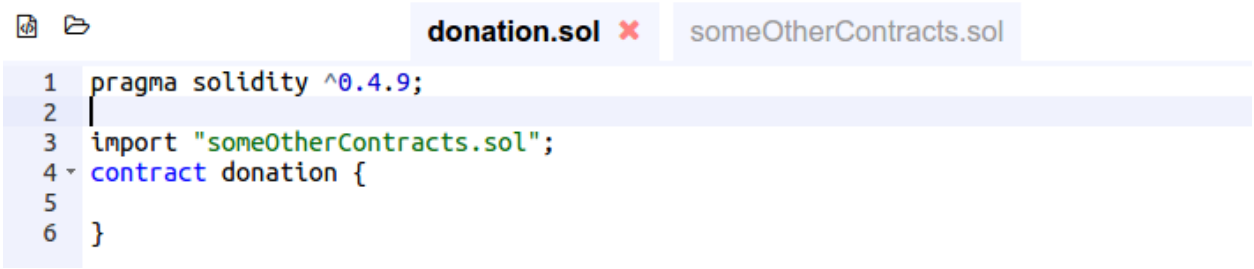
For a tutorial about importing files see this tutorial.

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

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

14.1 Importing a file from the browser's local storage

Files in Remix can be imported just by specifying their path. Please use ./ for relative paths to increase portability.



14.2 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.

14.3 Importing from GitHub

It is possible to import files directly from GitHub with URLs like `https://github.com/<owner>/<repo>/<path to the file>`.

```
1  pragma solidity ^0.4.9;
2
3  import "http://github.com/ethereum/dapp-bin/standardized_contract_apis/datafeed.sol";
4
5  contract donation {
6
7  }
```

14.4 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.

CHAPTER 15

Remixd: Get access your local filesystem

`remixd` is an npm module. Its purpose is to give the remix web application access to a folder on your local computer.

The code of `remixd` is here .

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

You can install it just in the directory of your choice using this command: `npm install remixd`

Then `remixd -s <absolute-path-to-the-shared-folder> --remix-ide <your-remix-ide-URL-instance>` will start `remixd` and will share the given folder.

For example, to sync your local folder to the official Remix IDE, `remixd -s <absolute-path-to-the-shared-folder> --remix-ide https://remix.ethereum.org`

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.

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.

From `Remix IDE`, in the Plugin Manager you need to activate the remixd plugin.

A modal dialog 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.

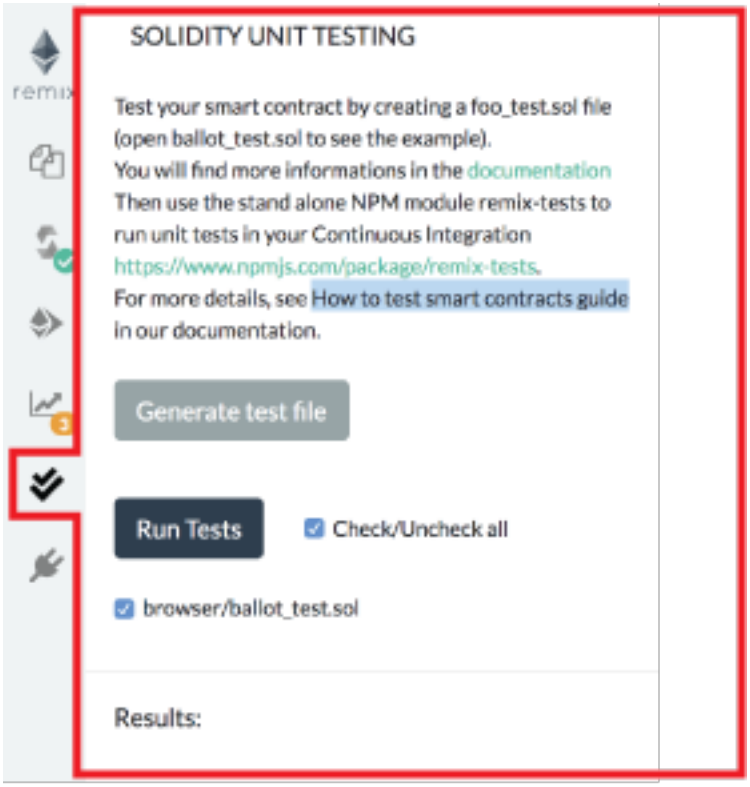
Click the File Explorers icon and in the swap panel you should now see the folder for `localhost`.

Click on the `localhost connection` icon:

CHAPTER 16

Unit Testing

Click the “double check” icon to get to the unit testing plugin. If you don’t see this icon, go to the plugin manager and load up the unit testing plugin.



16.1 Generate test File

This create a new solidity file in the current folder suffixed with `_test`. This file contains the minimun you need for running unit testing.

16.2 Run Tests

This execute tests. The execution is run in a separate environment and the result is displayed below.

| Available functions | Supported types |

| ———————- | ———————- |

| `Assert.ok()` | `bool` |

| `Assert.equal()` | `uint`, `int`, `bool`, `address`, `bytes32`, `string` |

| `Assert.notEqual()` | `uint`, `int`, `bool`, `address`, `bytes32`, `string` |

| `Assert.greaterThan()` | `uint`, `int` |

| `Assert.lesserThan()` | `uint`, `int` |

see https://github.com/ethereum/remix/blob/master/remix-tests/tests/examples_4/SafeMath_test.sol for some code sample

16.3 Continuous integration

remix-tests is also a CLI, it can be used in a continuous integration environement which support node.js. Please find more information in the remix-test repository

See also: example Su Squares contract and Travis build that uses remix-tests for continuous integration testing.

CHAPTER 17

Remix Plugin

The best documentation about how to build a plugin is currently in the readme of remix-plugin repo. Please go here to learn all about it.

CHAPTER 18

Finding Remix

So if you've found the documentation to Remix but don't know where to find Remix or if you want to run the remix-ide locally and want to find out where to download it - this page is here to help.

- 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.
- npm `remix-ide` package `npm install remix-ide -g`. `remix-ide` create a new instance of `Remix IDE` available at http://127.0.0.1:8080 and make the current folder available to Remix IDE by automatically starting `remixd`. see Connection to `remixd` for more information about sharing local file with `Remix IDE`.
- Github release: https://github.com/ethereum/remix-ide/releases . The source code is packaged at every release but still need to be built using `npm run build`.

CHAPTER 19

Remix 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 20

Code contribution guide

Remix is an open source tool and we encourage anyone to help us improve our tool. You can do that by opening issues, giving feedback or by contributing a pull request to our codebase.

The Remix application is built with JavaScript and it doesn't use any framework. We only rely on selected set of npm modules, like `yo-yo`, `csjs-inject` and 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 21

Community

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.

CHAPTER 22

Support chat

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 chat where we and other users try to answer your questions if you get stuck using Remix. Please, join the Remix channel and ask the community for help.

For anyone who is interested in developing a custom plugin for Remix or who wants to contribute to the codebase, we've opened another channel specially for developers working on Remix tool.